NECパーソナルコンピュータ
PC-9800シリーズ

NEC

Software Library

MS-DOS™3.3C
プログラム開発ツールマニュアル



Software Library

MS-DOS™3.3C
プログラム開発ツールマニュアル

ご注意

(1) 本書の内容の一部又は全部を無断転載することは禁止されています.
(2) 本書の内容に関しては将来予告なしに変更することがあります.
(3) 本書は内容について万全を期して作成いたしましたが，万一御不審な点や誤り，記載もれなどお気付きのことがありましたら御連絡下さい.
(4) 運用した結果の影響について(3)項にかかわらず責任を負いかねますので御了承下さい.

Microsoft(マイクロソフト)のロゴは米国マイクロソフト社の商標です.
MS-DOSは米国マイクロソフト社の商標です.
Intel および ASM86 は，米国インテル社の商標です.
386は米国インテル社の商標です.

輸出する際の注意事項

本製品（ソフトウェア）は，外国為替および外国貿易管理法の規定により，戦略物資等輸出規制品に該当します．従って，日本国外に持出す際には日本国政府の輸出許可申請等必要な手続きをお取り下さい.

AH064A

目　次

第1章　はじめに

第2章　LINK：リンカ

第3章 SYMDEB：シンボリックデバッグユーティリティ

第4章 LIB：ライブラリマネージャ

第5章 MAKE：プログラムメインテナ

第1章 はじめに

本書〝プログラム開発ツールマニュアル〟では，マクロアセンブリ言語によるプログラム開発を中心に，そこで用いる各種ユーティリティの解説を行っています．マイクロソフトマクロアセンブラ：MASM の文法などの詳細な解説は〝マクロアセンブラ ユーザーズ／リファレンスマニュアル（マクロアセンブラパッケージ添付）〟で行っています．

本章では，プログラム開発に関係するファイルの種類，本書で用いている表記法などの解説を行っています．

1.1 ファイル構成

本書で解説するプログラム開発に関連するユーティリティとしては，つぎのようなファイルがあります．

LINK.EXE	リンカユーティリティ
SYMDEB.EXE	シンボリックデバッガ
MAPSYM.EXE	シンボルマップユーティリティ
LIB.EXE	ライブラリマネージャ
MAKE.EXE	プログラムメインテナ

本書では，上記の各ユーティリティプログラムについて，用途・使用方法などを扱っています．

1.2 MASMの利用にあたって

マイクロソフトマクロアセンブラ(MASM)は，8086／286／386マイクロプロセッサファミリーのための，マクロアセンブリ言語です．

マクロアセンブラでプログラムを開発するには，マクロアセンブリ言語の正しい文法とソースファイルの書式，それを記述するエディタの操作方法に関する予備知識を必要とします．また，8086／286／386マイクロプロセッサファミリーの機能と命令（インストラクション）セットに関する知識も必要です．

ユーザーは，MASMを用いることにより，8086マイクロプロセッサのセグメントアーキテクチャを理想的に論理的に扱うことができます．このMASMの文法，ディレクティブ，オペランド，用例などは，〝マクロアセンブラ ユーザーズ／リファレンスマニュアル〟を参照してください．

マイクロソフトマクロアセンブラは，8086／286／386マイクロプロセッサファミリーの，すべての命令（インストラクション）セットをサポートしています．したがって，8088，8086，80286，386のマイクロプロセッサ，およびオプションで8087，80287数値演算プロセッサを使用するコンピュータについてアセンブリプログラムを開発することができます．〝マクロアセンブラ ユーザーズ／リファレンスマニュアル〟には，ユーザーの便のために，これらのプロセッサの，すべての命令(インストラクション)の書式と機能を掲載しています．

注意

マイクロソフトマクロアセンブラ（MASM）とマクロアセンブラ ユーザーズ／リファレンスマニュアルは，別売りのマクロアセンブラパッケージに添付されています．

1.3 開発手順とユーティリティの用途

アセンブリプログラムのソースファイルの作成から，実際に動く（実行可能な）プログラムが完成するまでの手順は，つぎのような4段階に分けて考えることができます．

1. エディタによって，アセンブリプログラムのソースファイルを作る．
2. MASM（マクロアセンブラ）によって，ソースプログラムをアセンブルする．
3. LINK(リンカ)によって，別に開発していたルーチンやライブラリファイルのルーチンと結合し，実行可能なプログラムを生成する．
4. SYMDEB（シンボリックデバッガ）によって，プログラムを検証・テストする．

前記の他に用意されたユーティリティは，つぎのような用途に用います．

MAKE(プログラムメインテナ)を用いると，これらの4段階のそれぞれで行うコマンド入力を一括して，自動化することができます．

MAPSYMは，SYMDEBでデバッグするときに用いる，シンボルマップファイルを作成するときに用います．

CREF(クロスリファレンサ)は，デバッグ処理の助けとなるもので，プログラム中のすべてのシンボルのクロスリファレンスリストを作成します．

LIB(ライブラリマネージャ)は，リンクの際に用いるライブラリファイルを管理するもので，ユーザー独自のライブラリファイルを作成したり，ライブラリファイルの内容や構造を変更するときに用います．

最後のテストを終了したプログラムは，MS-DOSコマンドとして，いつでも利用することができます．ユーザーが開発したプログラムも，MS-DOSで供給されるプログラム（コマンド）と同様に，コマンドパラメータを受け付けることができ，バッチファイルの中でも利用することができます．

1.4　本書で用いる表記法

このマニュアルでは，つぎのような表記法を用いています．

表　記	意　味
CAPS	**大文字のアルファベット**は，表記されているとおりに入力する，コマンド名，パラメータなどを表します．なお通常，大文字／小文字はどちらでも同様に扱われます．
〈　〉	**山形カッコ**は，その位置に入力する項目を表します．たとえば，<ファイル名>というところでは，ユーザーの作成したファイルの名前を入力します．
[　]	**角形カッコ**は，入力を省略できる項目を表します．省略した場合は，システムであらかじめ設定していた処理が行われます．
|	**縦線**は，選択項目の区切りを表します．この記号で区切られた項目の中から，必要なものを選んで入力することを表します．
…	**繰り返し記号**．必要に応じて，項目を繰り返し入力することを表します．
□	枠で囲まれた文字は，キーボード上の特定のキーを示します．たとえば，CTRLはコントロールキーを表します．また，CTRL+Cのような表記は，CTRLキーを押しながらCキーも押すことを表します．

カンマ（,），コロン（：），セミコロン（；），スラッシュ（／），イコール記号（＝）などの記号類とスペース（空白）は，コマンドなどの書式（文法）の一部です．表記されている位置に，正確に入力してください．なお，省略できる項目は，該当箇所で解説されています．

第2章
LINK：リンカ

2.1 イントロダクション

マイクロソフトリンカ：LINK は，マクロアセンブラ MASM または C, Pascsl のような高級言語コンパイラによって生成されたオブジェクトコードから，実行可能プログラムを作成します．LINK は，この結果生成されたプログラムを実行可能（.EXE）出力ファイルにコピーします．

ユーザーは，MS-DOS コマンド行上でこのファイルの名前をタイプすることよって，このプログラムを実行することができます．

LINK を使用するために，1 つまたそれ以上のオブジェクトファイルを作成し，つぎにこれらのファイルをすべての必要とされるライブラライリファイルと一緒に指定し，処理のためリンクしなければなりません．

LINK はオブジェクトファイル内のコードおよびデータを結合し，指定されたライブラリを探索し，ルーチンおよび変数に対する外部参照を解決します．

つぎに，LINK は，リロケータブル（再配置可能）な実行イメージおよび，リロケート（再配置）情報を，実行可能ファイルにコピーします．

MS-DOS は，LINK が生成したリロケート情報を使用して，任意の都合のよいメモリアドレスに実行可能イメージをロードし実行します．

LINK は，最高 1 メガバイトまでのコードおよびデータが入っているプログラムを，処理することができます．

本章では，実行可能なプログラムを作成するための LINK の使用方法が説明されています．ここでは，リンク処理を制御するために LINK コマンド行中で使用可能なスイッチも，定義されています．この章の最後のセクションでは，LINK がプログラムを作成する方法について説明しています．

2.2 LINKの起動と使用方法

本節では，実行可能プログラムを作成するためにLINKを起動し，使用する方法を説明しています．3つの異なった方法によって，すなわち，MS-DOSコマンド行を通して，プロンプトに応答して，または応答ファイルを使用して，LINKを使用することができます．

いったんLINKが開始されると，LINKは，ユーザーか指定したファイルを処理するか，または他のファイルの入力を促します．CTRL＋Cキーを押すことによって，いつでもLINKを中止することができます．

2.2.1 コマンドラインによる方法

"LINK"およびその後に処理したいファイルの名前をタイプすることによって，実行可能プログラムを作成することができます．このコマンドの書式は，つぎのように指定されます．

LINK　[〈スイッチ〉・・・]〈オブジェクトファイル名〉・・・，[〈出力ファイル名〉]，[〈マップファイル名〉]，[〈ライブラリファイル名〉・・・]

〈スイッチ〉は，LINKの動作を制御するもので，必要に応じて指定します．

〈オブジェクトファイル名〉は，一緒にリンクしたいオブジェクトファイルの名前です．これらのファイルは，MASMまたは高水準言語のコンパイラを使用して，作成されなければなりません．このファイル名は省略することはできません．

〈出力ファイル名〉は，作成したい実行可能ファイルの名前です．出力ファイル名の指定を省略すると，LINKは，最初のオブジェクトファイル名の拡張子を".EXE"に変えたファイル名を自動的に作成します．

〈マップファイル名〉は，マップリスティングを受け取るファイルの名前です．ファイルの名前が指定されず，／MAP，／LINENUMBERSスイッチも指定されない場合，マップファイルは作成されません．

〈ライブラリファイル名〉は，プログラムにリンクしたいルーチン群のはいっているライブラリの名前です．ライブラリ名を指定しない場合は，セミコロン（；）をタイプしてください．

特定のファイル名が指定されない場合でも，そのファイル名を区切るカンマは必要です．

複数のオブジェクトファイル名，ライブライリファイル名を指定するときは，これらの名前をスペース（空白）またはプラス記号（＋）で区切りながら並べます．

スイッチは，コマンド行上のどこにでも入れることができます．

LINKでは，コマンド行中に最低1つのオブジェクトファイル名が必要です．出力ファイル名が指定されない場合，LINKは，この行中で検出した先頭のオブジェクトファイルのファイル名を使用し，これに".EXE"というファイル名拡張子を付加することによって，省略時の出力ファイル名を作成します．

拡張子が指定されなかった場合，LINKはこれらのファイルのための省略時のファイル拡張子を使用します．LINKは，オブジェクトファイルに対して〝.OBJ″を，実行可能ファイルに対して〝.EXE″をマップファイルに対して〝.MAP″を，またライブラリファイルに対して〝.LIB″を使用します．

例：

```
LINK file.obj, file.exe, file.map, file.lib
```

この例では，〝file.obj″というオブジェクトファイルを使用して，〝file.exe″という実行可能ファイルが作成されます．LINKは，このプログラム中で使用されているルーチンおよび変数のための〝file.lib″というライブラリを探索します．LINKは，プログラム中のセグメントおよびグループのリストが入っている〝file.map″という名前のマップファイルを作成します．

```
LINK startup + file,file,file,;
```

この例では〝startup.obj″および〝file.obj″という2つのオブジェクトファイルから，〝file.exe″という名前の実行可能ファイルが作成されます．LINKはマップファイルを作成しますが，どのライブラリも探索しません．

```
LINK  moda+modb+modc+startup/PAUSE,, abc/MAP,  ¥lib¥math
```

この例では〝moda.obj″，〝modb.obj″，〝modc.obj″および〝startup.obj″という4つのオブジェクトモジュールがリンクされ，このプログラム中で使用されているルーチンおよびデータのための〝math.lib″というライブラリファイル（¥libディレクトリ内）が探索されます．次に，〝moda.exe″という名前の実行可能ファイルおよび〝abc.map″という名前のマップファイルが作成されます．なお／PAUSEスイッチが指定されているのでLINKは実行可能ファイルを作成する前に休止します．

2.2.2 コマンドプロンプトによる方法

リンカのコマンド名〝LINK″のみをタイプすると，LINKは必要な情報の入力を求めるコマンドプロンプトを表示します．この方法の手順は，つぎのようになります．

1. つぎのようにタイプし，リターンキーを押します．

```
LINK
```

LINKは，つぎのコマンドプロンプトによって，ユーザーがリンクしたいオブジェクトファイルの入力を求めます．

```
Object Modules [.OBJ]:
```

2. リンクしたいオブジェクトファイルの名前を，タイプします．ファイル名拡張子が指定されなかった場合，LINK は省略時処理で〝.OBJ〟を使用します．
複数の名前を指定する場合は，スペース（空白）またはプラス記号（＋）で区切りながら並べます．
指定する名前が1行上に入り切れない場合，その行の最後にプラス記号（＋）をタイプしてリターンキーを押します．LINK は，他のオブジェクトファイルの入力を促すコマンドプロンプトを表示します．

　すべてのオブジェクトファイル名を指定したら，リターンキーを押します．LINK は，つぎの出力ファイル名の指定を求める，第2のコマンドプロンプトを表示します．

Run File [filename.EXE]:

3. 作成したい実行可能ファイルの名前をタイプし，リターンキーを押します．拡張子が指定されなかった場合，LINK は省略時処理で〝.EXE〟を使用します．省略時の実行可能ファイル名を使用したい場合は，単にリターンキーを押す．このファイル名は先頭のオブジェクトファイルと同一ですが，このファイルの拡張子は〝.EXE〟になります．
　リターンキーが押されると，LINK は，リストマップファイル名の入力を求める，第3のコマンドプロンプトを表示します．

List File [NUL.MAP]:

4. 作成したいリストマップファイルの名前をタイプしてリターンキーを押します．ファイル名拡張子が指定されない場合，LINK は省略時処理で〝.MAP〟を使用します．マップファイルを希望しない場合は，ファイル名をタイプせずに，リターンキーを押します．
　リターンキーが押されると，LINK は，リンク処理に用いるライブラリファイル名の入力を求める最後のコマンドプロンプトを表示します．

Libraries [.LIB]:

5. 参照されているが，プログラムで定義されていないルーチンまたは変数が入っているすべてのライブラリの名前をタイプします．
　複数のライブラリファイル名を指定する場合は，これらの名前をスペース（空白）またはプラス記号（＋）で区切りながらタイプします．ファイル名拡張子が指定されなかった場合，LINK は省略時解釈で〝.LIB〟を使用します．指定する名前が1行上に入り切らない場合は，この行の最後にプラス記号（＋）をタイプして，リターンキーを押します．LINK はさらに他のファイル名の入力を求めます．

すべての名前を入力した後，リターンキーを押します.

どのライブラリも検索しない場合は，単にリターンキーを押します.

最後のコマンドプロンプトに対する入力が済むと，LINKは，ただちに処理を開始します.

注意

ファイルが他のディレクトリ内または他のディスク上にある場合は，パス名やドライブ名を含めた正しいファイル名を指定します. LINKは，オブジェクトファイルを見つけることができない場合，メッセージを表示し，必要ならディスクを変更できるように待ち状態になります.

入力行の終わりにスイッチを付加することができます.

任意のプロンプトの後にセミコロン(;)をタイプすることによって，それ以降のすべてのプロンプトにおいて省略時の処理を選択するように，LINKに命令することができます. LINKはセミコロンを検出すると初期設定に従った処理を行い実行可能ファイルを作成します.

例:

コマンドプロンプトを利用した入力例をつぎに示します.

```
LINK

Object  Modules  [.OBJ]: moda + modb +
Object  Modules  [.OBJ]: modc + startup/PAUSE
Run  File  [moda.EXE]:
List  File  [NUL.MAP] : abc
Libraries  [.LIB]: ¥lib¥math
```

この例では"moda.obj"，"modb.obj"，"modc.obj"および"startup.obj"というオブジェクトモジュールがリンクされ，プログラム中で使用されているルーチンおよびデータのために"math.lib"（¥libディレクトリ内）というライブラリファイルが検索されます. 次に"moda.exe"という名前の実行可能ファイルおよび"abc.map"という名前のマップファイルが作成されます. オブジェクトファイルのプロンプト行中で／PAUSEスイッチが指定されることによって，実行可能ファイルを作成する前に，LINKが休止します.

2.2.3 応答ファイルによる方法

応答ファイル内に，処理に関係したすべてのファイルの名前をリストすることによって，またLINKコマンド行上でこの応答ファイルの名前を指定することによって，プログラムを作成することができます．このコマンド行は，つぎのように指定されます．

LINK　@〈応答ファイル名〉

〈応答ファイル名〉は，LINKが使用するファイル名を収めた応答ファイル名です．応答ファイル名を用いるときは，この前にアットマーク（@）が置かれなければなりません．このファイルが他のディレクトリ内または他のディスクドライブ上にある場合，これにパス名またはドライブ名がつけられなければなりません．

応答ファイルには，好きなファイルの名前をつけることができます．ファイルの内容は，つぎのような形式で構成されています．

〈オブジェクトファイル名〉
[〈出力ファイル名〉]
[〈マップファイル名〉]
[〈ライブラリファイル名〉]

各グループのファイル名は，別々の行上で指定されなければなりません．応答ファイル名の各行が，前項で解説したコマンドプロンプトに対する入力に対応しています．指定する名前が1行に入り切らない場合は，その行の最後にプラス記号（+）をタイプすることによって，次の行に継続することができます．あるグループに対してファイル名を指定しない場合は，その行を空にしておかなければなりません．スイッチは，どの行でも指定可能です．

応答ファイル内のどの行上でも，セミコロン（;）を入れることができます．LINKは，セミコロンを検出すると応答ファイル内でまた指定されていないすべてのファイルに対して，省略時のファイル名を自動的に使用し，ファイルの残りの部分は無視されます．

応答ファイルを使用してリンク処理を行うとき，LINKは，画面上に応答ファイルからの各応答を表示し処理を進めます．応答ファイルに，必要なファイル名前が入っていない場合，LINKは不足しているファイル名の入力を求めるコマンドプロンプトを表示し，応答がキーボードから入力されるまで待ち状態になります．

例：

```
moda modb modc startup
／PAUSE
abc
¥lib¥math
```

この応答ファイルは，〝moda〞，〝modb〞，〝modc〞および〝startup〞という4つのオブジェクトモジュールをリンクするように，LINKに指示するものです．LINKは，〝moda.exe〞という実行可能ファイルを作成する前に，ユーザーがディスクを交換できるように，休止します．LINKは，〝abc.map〞というマップファイルも作成し，〝¥lib¥math.lib〞というライブラリを検索します．

2.2.4 ライブラリファイルの検索パスの設定

コマンド内で，ライブラリ名と一緒に1つまたはそれ以上の検索パスを指定することによって，あるいはLINKを呼び出す前に環境変数〝LIB〞に検索パスを割り当てることによって，これらのライブラリを見つけるためにディレクトリおよびディスクドライブを検索するようにLINKに命令することができます．

検索パスとは，ディレクトリまたはドライブ名の単なるパス指定のことです．検索パスは，LINKコマンド行上でライブラリ名と一緒に，または〝Libraries〞コマンドプロンプトに応答して，入力されます．最高16までの検索パスを指定することができます．また，MS-DOSのSETコマンドを使用して，検索パスをLIB変数に割り当てることもできます．この場合，検索パスはセミコロン（;）によって区切られなければなりません．

検索パスは，ライブラリに明示のパス指定またはドライブ名がない場合のみ使用されます．LINKは，最初に，カレントディレクトリを，次に，このコマンドで指定された各々のディレクトリおよびディスクドライブを，最後に，LIB変数で指定されたディレクトリおよびディスクドライブを検索します．LINKは指定された名前のライブラリを検出するまで，または検索すべき場所がなくなるまで，検索を続行します．

ディレクトリおよびディスクドライブは，検索パスがコマンド中で指定された順に，またはLIBに割り当てられた順に検索されます．ライブラリ名に対して，パス指定またはドライブ名が指定されている場合，LINKは指定されたディレクトリまたはディスクドライブにあるライブラリのみを検索します．

例：

```
LINK file, , file.map, A:¥altlib¥math.lib + common + B: + D:¥lib¥
```

この例では，LINKは，〝math.lib〞というライブラリを見つけるために，ドライブA上の〝¥altlib〞ディレクトリのみを検索しますが，〝common.lib〞を見つけるために，ドライブA上のカレントディレクトリ，ドライブB上のカレントディレクトリ，および最後にドライブD上の〝¥lib〞というディレクトリを検索します．

```
SET LIB = C:¥lib; U:¥system¥lib
LINK file, , file. map, math + common
```

この例では，LINKは，〝math.lib″および〝common.lib″というライブラリを見つけるために，カレントディレクトリ，ドライブC上の〝¥lib″というディレクトリ，およびドライブU上の〝¥system¥lib″というディレクトリを検索します．

2.2.5 マップファイル

マップファイルには，プログラム中のすべてのセグメントの名前，ロードアドレス，および長さがリストされます．このファイルには，このプログラム中のすべてのグループの名前とロードアドレスプログラム開始アドレス，および検出される可能性のあるすべてのエラーについてのメッセージもリストされます．

LINKコマンド行で／MAPスイッチが指定された場合，このマップファイルには，すべてのパブリックシンボルの名前およびロードアドレスがリストされます．

セグメント情報はつぎのような形式で出力されます．

Start	Stop	Length	Name	Clase
00000H	0172CH	0172DH	TEXT	CODE
01730H	01E19H	006EAH	DATA	DATA

〝Start″および〝stop″の項には，各セグメントの先頭および最終バイトの20ビットアドレス(16進)が示されます．これらのアドレスは，アドレス0000Hと想定されるロードモジュールの開始点からの相対アドレスです．オペレーティングシステムは，このプログラムが実際にロードされるとき，それ自身の開始アドレスを設定します．〝Length″の項には，セグメントの長さがバイト単位で〝Name″の項には，セグメントの名前が，〝Class″の項にはセグメントのクラス名が示されます．

グループ情報は，つぎのような形式で出力されます．

Origin	Group
0000:0	IGROUP
0173:0	DGROUP

この例では，IGROUPはコード(命令)グループの名前，DGROUPはデータグループの名前です．

リスティングファイルの終わりに，LINKはプログラムのエントリポイントのアドレスを示します．

LINKコマンド行で／MAPスイッチが指定された場合，LINKはパブリックシンボルリストをマップファイルに付加します．これらのシンボルは，2回，すなわち1回目はアルファベット順に，次のロードアドレス順に示されます．このリストは，つぎのような形式で出力されます．

```
ADDRESS          PUBLICS BY NAME
0000:1567    brk
0000:1696    chmod
0000:01DB    chkstk
0000:131C    clearerr
0173:0035    fac

ADRESS           PUBLICS BY VALUE
0000:01DB    chkstk
0000:131C    clearerr
0000:1567    brk
0000:1696    chmod
0000:0035    fac
```

パブリックシンボルのアドレスは，セグメント：オフセットという形式で示されます．これらは，シンボルの，ロードモジュールの開始点からの相対位置を示しています．このモジュールの開始点は，アドレス0000:0000にあると想定されます．

／HIGHおよび／DSALLOCATEスイッチが指定され，このプログラムのコードおよびデータをリンクしたものが64Kバイト以下であるとき，マップファイルには，通常大きなセグメントアドレスを持つシンボルを示すことができます．これらのアドレスは，シンボルが，プログラムコードおよびデータの実際の開始点より低い位置にあることを示しています．たとえば，つぎのシンボルエントリでは，"template"はこのプログラムの開始点より低い位置にあることが示されています．

```
FFF0:0A20       template
```

templateの20ビットアドレスは，00920Hであることに注意してください．

2.2.6　一時ディスクファイル —— VM．TMP

LINKは，通常，リンク処理のために使用可能なメモリを使用します．LINKは，使用可能なメモリを使い尽くした場合，カレントディレクトリ内に"VM．TMP"という名前の一時ディスクファイルを作成します．LINKは，このファイルを作成したとき，つぎのメッセージを表示します．

```
VM.TMP has been created.
Do not change diskette in drive, <d:>
```

このメッセージが出力された後は，リンク処理が終了するまで，このドライブからディスクを取り外してはいけません．LINKは，実行可能ファイルを作成した後，一時ファイルを自動的に削除します．

警告

ユーザー自身のファイルのために，VM．TMPというファイル名を使用してはなりません．LINKが一時ファイルを作成したとき，以前に同一の名前が付けられているすべてのファイルを破壊します．

2.3　LINKのスイッチ

リンカスイッチによって，LINKによって実行されるタスクが指定され，制御されます．すべてのスイッチは，リンカスイッチ文字である，スラッシュ（／）で開始します．スイッチは，LINKコマンド行上のどこにでも，指定可能です．
LINKにはつぎのようなスイッチがあります．

／PAUSE	リンク中に休止
／MAP	パブリックシンボルマップの作成
／STACK	スタックサイズの設定
／CPARMAXALLOC	最大割り当てスペースの設定
／HIGH	上位開始アドレスの設定
／DSALLOCATE	データグループの割り当て
／OVERLAYINTERRUPT	オーバーレイ割り込みの設定
／LINENUMBERS	行番号の表示
／NOIGNORECASE	大文字小文字の区別
／NOGROUPASSOCIATION	グループを取り除く
／NODEFAULTLIBRARYSEARCH	省略時のライブラリを無視する
／SEGMENTS	最大セグメントの設定
／DOSSEG	DOSのセグメント配列の利用

2.3.1 リンク中に休止する

書 式 ／PAUSE

省略形 ／P

／PAUSEスイッチが指定された場合，LINKは実行可能ファイルをディスクに書き込む前に休止します．このスイッチによって，LINKが実行可能(.EXE)ファイルを出力する前に，ユーザーはディスクを交換することができます．

／PAUSEスイッチが指定された場合，LINKは，実行ファイルを作成する前につぎのメッセージを表示します．

```
About to generate.EXE file
Change diskette in drive <d:> and press <ENTER>
```

LINKは,リターンキーが押されると,処理を再開します．／PAUSEスイッチが指定されなかったとき，LINKはリンク処理を最初から終わりまで，停止しないで実行します．

注意

VM.TMPファイルが作成されている場合，このファイルのために使用されているディスクを取り外してはなりません．

例:

```
LINK file.obj／PAUSE, file.exe, , ¥lib¥math.lib
```

このコマンドによって，LINKは，“file.exe”という実行可能ファイルを作成する直前に，休止します．実行可能ファイルが作成された後，MASMは，ユーザーがもとのディスクを戻すことができるように，再び休止します．

2.3.2 パブリックシンボルマップの作成

書 式 ／MAP

省略形 ／M

／MAPスイッチが指定された場合，LINKは，プログラムで宣言されたすべてのパブリックシンボルのリスティングを作成します．このリストは，LINKによって作成されたマップファイルにコピーされます．リスティングファイルの書式についての詳細は，2.2.5の“マップファイル”を参照してください．

注意

LINKコマンドでマップファイルを指定しなかった場合，"List file"プロンプトで，またはこのプロンプトの前に，／MAPスイッチを入力することによって，LINKにマップファイルを作成されることができます．LINKは，作成されたマップファイルにこのコマンド中で指定された先頭のオブジェクトファイルと同一のファイル名，および".MAP"という省略時の拡張子をつけます．

例：

```
LINK file.obj, file.exe, file.map／MAP,;
```

このコマンドは，"file.obj"というファイル内のすべての共有記号のマップを作成します．

2.3.3　スタックの大きさのセット

書　式　／STACK：〈サイズ〉

省略形　／ST

／STACKスイッチが指定された場合，プログラムのスタックが〈サイズ〉によって指定されたバイト数にセットされます．サイズは，1～65535の範囲内にある任意の正の整数値が可能です．この値は，10進，8進，または16進数が可能です．8進数はゼロで，16進数は"0x"で始められなければなりません．

LINKは，通常，プログラムのスタックの大きさをオブジェクトファイルで示されているすべてのスタックセグメントの大きさに基づいて自動的に計算します．／STACKが指定された場合，LINKは値を計算する代わりに，指定されたサイズを使用します．

このプログラム中にスタックセグメントが存在しない場合．LINKはエラーメッセージを表示します．このメッセージが出力されないようにするために，すべてのプログラムにおいて最低1つのスタックセグメントを定義しなければなりません．

例：

```
LINK file.obj／STACK:512, file.exe,,;
```

この例では，スタックの大きさは512バイトにセットされます．

```
LINK moda+modb, run／ST:0xFF,ab.map,¥lib¥start;
```

この例では，スタックの大きさは255（FFH）バイトにセットされます．

```
LINK startup+file／ST:030,file,,;
```

この例では，スタックの大きさは24（8進の30）バイトにセットされます．

2.3.4 最大割り当てスペースのセット

書　式　／CPARMAXALLOC：〈サイズ〉

省略形　／C

／CPARMAXALLOCスイッチを指定することによって，プログラムがメモリ中にロードされたとき，このプログラムが必要な，最大の16バイトのパラグラフ数がセットされます．この数は，プログラムをロードする前にこのプログラムのためにスペースを振り当てるとき，オペレーティングシステムによって使用されます．

〈サイズ〉は，1～65535の範囲内にある任意の整数値が可能です．これは，10進，8進，または16進数でなければなりません．8進数はゼロで，16進値は"0x"で始められなければなりません．

LINKは，通常，最大のパラグラフ数を65535にセットします．これはすべてのメモリを表わしているので，オペレーティングシステムは常にこの要求を拒否し，存在する最大の連続したブロックのメモリを割り当てます．／CPAMAXALLOCスイッチが指定された場合，オペレーティングシステムはこのスイッチで指定された量のスペースを割り当てます．これは，メモリ中の他のスペースを他のプログラムが使用できることを意味します．

〈サイズ〉が，プログラムによって必要とされる最小のパラグラフ数より小さい場合，LINKはユーザーの要求を無視し，最大の値が必要とされる最小のパラグラフ数と等しくなるようにセットします．プログラムによって必要とされる最小のパラグラフ数は，常にこのプログラム中のコードおよびデータのパラグラフ数以上になります．

例：

```
LINK file.obj/C:15, file.exe,,;
```

この例では，最大の割り当ては15パラグラフにセットされます．

```
LINK moda + modb, run/CPARMAXALLOC:0xff, ab.map,;
```

この例では，最大の割り当ては255（FFH）パラグラフにセットされます．

```
LINK startup + file/C:030,,;
```

この例では，最大の振当ては24（8進の30）パラグラフにセットされます．

2.3.5 上位開始アドレスのセット

書 式 ／HIGH

省略形 ／H

／HIGH スイッチが指定された場合，プログラムの開始アドレスが使用可能なメモリの中のできるだけ高いアドレスにセットされます．このスイッチによって，LINK は，オペレーティングシステムにこのプログラムをできるだけ高いアドレスにロードさせる情報を，実行可能ファイルに付加します．

／HIGH が指定されない場合，プログラムの開始アドレスはメモリ中のできるだけ低いアドレスにセットされます．

例：

```
LINK startup + file/HIGH, file, file, ;
```

この例では，"file.exe"内のプログラムの開始アドレスが，使用可能なメモリ中のできるだけ高いアドレスにセットされます．

2.3.6 データグループの割り当て

書 式 ／DSALLOCATE

省略形 ／D

／DSALLOCATE スイッチが指定された場合，"DGROUP"という名前のグループに属する項目にアドレスを割り当てるとき，通常の処理を逆にするように LINK に命令します．通常，LINK は，グループ内の最下位バイトにオフセット 0000H を割り当てますが，／DSALLOCATE が指定された場合，LINK は，このグループ内の最上位バイトにオフセット FFFFH を割り当てます．この結果，出力されたデータが，DGROUP が入っているメモリセグメント中のできるだけ高いアドレスにロードされます．

／DSALLOCATE スイッチは，プログラムの開始点より低いアドレスの未使用のメモリを利用するために，通常／HIGH スイッチといっしょに使用されます．LINK は，DGROUP 内のすべての使用可能なバイトは，このプログラムより低い隣接したアドレスのメモリを占有していると想定します．このグループを使用するために，セグメントレジスタは DGROUP の開始アドレスにセットされなければなりません．

例:

LINK startup + file／HIGH／DSALLOCATE, file, file,;

この例では，このプログラムをメモリ中のできるだけ高いアドレスにロードし，つぎにDGROUP内のすべてのデータ項目がこのグループ内のできるだけ高いアドレスにロードされるようにこれらの項目のオフセットを調整することを，LINKに命令します．

2.3.7 行番号の表示

書 式 ／LINENUMBERS

省略形 ／LI

／LINENUMBERSスイッチが指定された場合，各プログラムソース行の開始アドレスをリストするように，LINKに命令します．この開始アドレスは，実際には，対応するソース行を構成している命令群のアドレスです．LINKは，この情報をマップファイルにコピーします．ユーザーはプログラムのデバッギングのために，このファイル内の情報を使用することができます．

LINKは，LINKコマンド行中でのマップファイル名が指定された場合のみ，また指定されたオブジェクトファイルに行番号の情報がある場合のみ，行番号の番号付けを行います．行番号の番号付けは，高水準言語のコンパイラにおいてのみ，使用可能です．オブジェクトファイルに行番号情報がない場合，LINKは／LINEMUMBERSスイッチを無視します．

注意

LINKコマンド中にマップファイルを指定しなかった場合，〝List file〞プロンプトで，またはこのプロンプトの前に／LINENUMBERSスイッチを入力することによって，LINKにマップファイルを作成させることができます．LINKは，作成されたマップファイルに，このコマンド中で指定された先頭のオブジェクトファイルと同一のファイル名，および〝.MAP〞という省略時の拡張子をつけます．

例:

LINK file.obj／LINENUMBERS, file.exe, file.map,;

この例では，〝file.obj〞というオブジェクトファイル内の行番号情報が，〝file.map〞というマップファイル内にコピーされます．

2.3.8　大文字小文字の区別

書　式　／NOIGNORECASE

省略形　／NOI

／NOIGNORECASEスイッチが指定された場合，シンボル名における大文字と小文字を区別して扱うように，LINKに命令します．通常，LINKは大文字と小文字を同一のものと見なし，"TWO"，"two"および"Two"という名前を同一のシンボルとして扱います．／NOIGNORECARE"を指定することによって，LINKはこれらの名前を一意な記号として扱います．

／NOIGNORECAREスイッチは，通常，高級言語のコンパイラによって作成されたオブジェクトファイルと一緒に使用されます．コンパイラの中には，大文字と小文字を区別のある英字として扱い，LINKもそうであると想定するものもあります．

例：

```
LINK file.obj/NOI, file.exe,file.map, ¥lib¥slibc.lib;
```

このコマンドによって，LINKは記号名における大文字と小文字を区別のある英字として扱います．"file.obj"というオブジェクトファイルは，"¥lib¥slibc.lib"という標準C言語ライブラリ内のルーチン群とリンクされます．C言語は，大文字と小文字が別々に扱われることを期待しています．

2.3.9　省略時のライブラリを無視する

書　式　／NODEFAULTLIBRARYSEARCH

省略形　／NOD

／NODEFAULTLIBRARYSEARCHスイッチが指定された場合，オブジェクトファイル内で検出される可能性のあるすべてのライブラリ名を無視するように，LINKに命令します．高級言語のコンパイラは，省略時の一群のライブラリが確実にこのプログラムにリンクされるように，ライブラリ名をオブジェクトファイルに付加することができます．このスイッチによって，これらの省略時のライブラリが無効になり，LINKコマンド行上に，希望するライブラリを明示で指定することができます．

例：

```
LINK startup + file/NOD, file.exe, ,¥lib¥math.lib
```

この例で，"startup.obj"および"file.obj"というオブジェクトファイルが，"¥lib¥math.lib"というライブラリのルーチン群にリンクされます．startup.objまたはfile.obj内で指定されていた可能性のある，すべての省略時のライブラリは無視されます．

2.3.10 プログラムからグループを取り除く

書 式 ／NOGROUPASSOCIATION

省略形 ／NOG

／NOGROUPASSOCIATION スイッチを指定した場合，データおよびコード項目にアドレスを割り当てるとき，グループの関連を無視するようにLINKに命令します．

参考

このスイッチの使用は，お勧めできません．

2.3.11 オーバーレイ割り込みのセット

書 式 ／OVERLAYINTERRUPT：〈値〉

省略形 ／O

／OVERLAYINTERRUPT スイッチが指定された場合，オーバーレイローディングルーチンの割り込み番号が〈値〉にセットされます．このスイッチによって，通常のオーバーレイ割り込み番号（03FH）が無効になります．

〈値〉は，0〜255の範囲内の任意の整数値が可能です．これは，10進，8進，または16進数でなければなりません．8進数は，先行ゼロで，16進数は，〝0x〞で始められなければいけません．

注意

標準MS-DOS割込みと矛盾する割り込み番号を使用することは，お勧めできません．

例：

```
LINK file.obj／O：255, file. exe, ,;
```

この例では，オーバーレイ割り込み番号が255にセットされます．

```
LINK moda＋modb, run／OVERLAY：0xff, ab. map,;
```

この例では，オーバーレイ割込み番号が255（FFH）にセットされます．

```
LINK startup＋file, file／0377, ,;
```

この例では，オーバーレイ割込みが255（8進の377）にセットされます．

2.3.12　最大のセグメント数のセット

書　式　／SEGMENTS：〈サイズ〉

省略形　／SE

／SEGMENTS スイッチが指定された場合，プログラムのセグメントの数〈サイズ〉の上限をLINKにわたします．LINKは，指定された限界より多く検出した場合，エラーメッセージを表示し停止します．このスイッチは，省略の限界である128のセグメントを無効にするために使用されます．

〈サイズ〉は，1～1024の範囲内の任意の整数値が可能です．これは，10進，8進，または16進数でなければなりません．8進数は先行ゼロで，16進数は〝0x″で始められなければなりません．

／SEGMENTS が指定されなかった場合，LINKは最高128のセグメントを処理するのに充分なメモリスペースを割り当てます．このプログラムが128より多くのセグメントによって構成されている場合，セグメントの限界をより大きな値にセットすることによってLINKが処理可能なセグメント数を増加させます．

例:

```
LINK file.obj/SE:10, file.exe, ,;
```

この例では，セグメントの限界が10にセットされます．

```
LINK moda + modb, run/SEGMENTS:0xff,ab.map,;
```

この例では，セグメントの限界が255（FFH）にセットされます．

```
LINK startup + file, file/SE:030, , ;
```

この例では．セグメントの限界が24（8進の30）にセットされます．

2.3.13 DOSのセグメント配列を利用する

書 式 ／DOSSEG

省略形 ／DO

／DOSSEGが指定された場合，LINKは，MS-DOSセグメントの順序付けの規約に従って，実行可能ファイル内のすべてのセグメントを配置します．この規約は，つぎのようなものです．

1. "CODE"というクラス名が付けられたすべてのセグメントは，実行可能ファイルの開始点に置かれる．
2. "DGROUP"という名前のグループに属さないすべてのセグメントは，"CODE"セグメントの直後に置かれる．
3. "DGROUP"に属するすべてのセグメントは，このファイルの終わりに置かれる．

例：

```
LINK startup + test／DOSSEG, file, math + common
```

このコマンドによって，LINKは，"file.exe"という名前の実行可能ファイルを作成します．このファイル内のセグメントは，MS-DOSセグメントの順序付けの規約に従って配置されます．"start.obj"と"test.obj"というオブジェクトファイル内のセグメント，および"math.lib"と"common.lib"というライブラリからコピーされたすべてのセグメントは，前述の指定された順に配置されます．

2.4　LINKの動作方法

LINKは，もとのソースファイルで示されている命令に従って，プログラムのコードとデータセグメントを連結することによって，実行可能ファイルを作成します．これらの連結されたセグメントは，〝実行可能イメージ″を形成し，これらのイメージは，実行のためにこのプログラムが呼び出されたとき，メモリ中に直接コピーされます．したがって，LINKがセグメントを実行可能ファイルにコピーする順序および方法は，これらのセグメントがメモリ中にロードされる順序および方法を定義します．

プログラムのセグメントをリンクする方法をLINKに命令するために，SEGMENTディレクティブを使用してセグメントの属性を指定するか，またはプログラム中でGROUPディレクティブを使用することによって，セグメントのグループを形成します．これらのディレクティブは，プログラム中のすべてのセグメント順序におよび相対開始アドレスを定義するグループの関連，クラス，位置合わせ，および結合タイプを定義します．コマンド行スイッチを通してユーザーが指定したすべての情報の他に，この情報が有効です．

本節では，セグメントを結合し，メモリ中の項目に対する参照を解決するための処理方法を説明します．

2.4.1　セグメントの位置合わせ

LINKは，セグメントの位置合わせのタイプを使用して，このセグメントの開始アドレスをセットします．位置合わせのタイプは，BYTE，WORD，PARA，およびPAGEです．これらは，バイト，ワード，パラグラフ，およびページの境界における開始アドレスと対応しており，それぞれ，1，2，16，および1024の倍数のアドレスを表わしています．

LINKは，セグメントを検出したとき，このセグメントを実行可能ファイルにコピーする前に，位置合わせのタイプをチェックします．位置合わせがWORD，PARA，またはPAGEである場合，LINKは，最後にコピーされたバイトが該当する境界で終了しているかどうか知るために，実行可能イメージをチェックし，そうでない場合，このイメージを余分なゼロバイトで埋めます．

2.4.2　フレーム番号

LINKは，プログラム中の各セグメントごとに，開始アドレスを計算します．この開始アドレスは，セグメントの位置合わせ，および実行可能ファイルにすでにコピーされたセグメントの大きさに基づきます．このアドレスは，オフセットおよび〝標準フレーム番号″によって構成されます．標準フレーム番号は，1バイトまたはそれ以上のバイトのセグメントが入っているメモリ中の先頭のパラグラフのアドレスを指定します．フレーム番号は，常に16の倍数(すなわち，パラグラフのアドレス）です．オフセットは，このパラグラフの開始点からセグメント

内の先頭バイトまでのバイト数です．BYTEおよびWORDの位置合わせの場合，オフセットは非ゼロの場合がありますが，PARAおよびPAGEの位置合わせの場合，常にゼロです．

セグメントをリンクするとき，このセグメントのフレーム番号をLINKによって作成されたマップファイルから取得することができます．フレーム番号は，このセグメントのために指定された〝開始〟アドレスの16進の先頭の5桁です．

2.4.3 セグメントの順序

LINKは，オブジェクトファイル内でセグメントを検出したときと同じ順序で，これらのセグメントを実行可能ファイルにコピーします．この順序は，LINKが同一のクラス名が付けられた複数のセグメントを検出しないかぎり，このプログラムを通して維持されます．同一のクラス名が付けられたセグメントは同一のクラスに属し，連結したブロックとして実行可能ファイルにコピーされます．

たとえば，以下のプログラム部分では，〝DATAX〟および〝DATAZ〟というセグメントは1クラスを形式します．両方のセグメントは，実行可能ファイルで〝TEXT〟セグメントの前にコピーされます．

```
DATAX     segment     'DATA'
DATAX     ends

TEXT      segment     'CODE'
TEXT      ends

DATAZ     segment     'DATA'
DATAX     ends
```

すべてのセグメントは，何らかのクラスに属します．クラス名が明示で定義されていないセグメントには，〝null〟クラス名が付けられ，nullクラス名が付けられた他のセグメントと一緒に連続したブロックとしてロードされます．

LINKでは，1クラス内のセグメント数や大きさは無制限です．1クラス内のすべてのセグメントを総計した大きさが，64Kバイト以上であっても構いません．

2.4.4 セグメントの結合

LINKは，結合タイプを使用して，同一のセグメント名を共有する複数のセグメントを単一の大規模なセグメントに結合すべきかどうかを決定します．結合タイプとは，public，stack，memory，common，privateがあります．

セグメントのタイプがpublicである場合，Linkは，自動的に，これを同一の名前が付けられ，同一のクラスに属する他のすべてのセグメントと結合します．LINKは，セグメントを結合するとき，確実に，これらのセグメントが連続しており，これらのセグメント内のすべてのアドレスが同一のフレームのアドレスからのオフセットを使用してアクセス可能になるようにします．この結果，このセグメントがソースファイル内で全体として定義されている場合と同一のものが生成されます．

LINKは，各々のセグメントの位置合わせのタイプを保存します．これは，これらのセグメントが単一の，大規模なセグメントに属していても，セグメント内のコードおよびデータはもとの位置合わせを保存するという意味です．結合されたセグメントが64Kバイトより大きい場合，LINKはエラーメッセージを表示します．

セグメントのタイプがstackまたはmenoryである場合，LINKはpublicセグメントと同様の結合操作を行います．唯一の例外は，stackセグメントの場合，LINKが最初のスタックポインタ値を実行可能ファイルにコピーすることということです．このスタックポインタ値は，検出された先頭のスタックセグメント（または結合されたスタックセグメント）の終わりまでのオフセットです．

セグメントのタイプがcommonである場合，LINKは，自動的に，これを同一の名前が付けられ，同一のクラスに属する他のすべてのセグメントと結合します．しかしLINKは，commonセグメントを結合するとき，各セグメントの開始点を同一のアドレスにロードし，一連のオーバラップしたセグメントを作成します．この結果，結合された最大のセグメントと同一の大きさの単一のセグメントが生成されます．

ソースファイル内で，セグメントのために明示の結合タイプが定義されていない場合のみ，このセグメントのタイプがprivateになります．LINKは，privateセグメントは結合しません．

2.4.5 グループ

グループによって，連続しておらず，同一のクラスに属していないセグメントは，同一のフレームアドレスからの相対アドレスで，アドレス指定することができます．

LINKは，グループを検出したとき，このグループ内の項目に対するすべてのメモリ参照が，同一のフレームアドレスからの相対アドレスになるように，これらの参照を調整します．LINKは，グループのすべての要素が同一の64Kバイトメモリ内に収まるかどうかをチェックしません．

グループ内のセグメントは，連続している必要はなく，同一のクラスに属する必要もなく，結合タイプが同一である必要もありません．グループ内のすべてのセグメントが，64Kバイト内に収まることだけが必要とされます．

グループは，セグメントがロードされる順序に影響を及ぼしません．クラス名を使用せず，オブジェクトファイルを正しい順序で入力しなかった場合，セグメントが連続しているという保障はありません．実際に，LINKは，同一の64Kバイトメモリ中にこのグループに属さないセグメントをロードする可能性があります．LINKは，グループ内のすべてのセグメントが64Kバイトのメモリ内に収まっていることを明示でチェックしませんが，この要件が満たされない場合，解決オーバフローエラーを検出する可能性があります．

2.4.6 参照の解決

いったん，プログラム中の各セグメントの開始アドレスが知られ，すべてのセグメントの組合せおよびグループが確立されてしまうと，LINKは，ラベルおよび変数にたいするすべての未解決の参照を〝解決〟することができます．未解決の参照を解決するために，LINKは該当するオフセットおよびセグメントのアドレスを計算し，アセンブラによって生成された一時的な値を新規の値で置き換えます．

LINKは，以下の4つの異なった参照を解決します．

- Short
- Near Self－Relative
- Near Segment－Relative
- Long

計算すべき値の大きさは，参照のタイプに依存します．LINKは，期待された大きさの参照においてエラーを検出した場合，未解決オーバフローメッセージを表示します．このことは，たとえば，プログラムが，16ビットのオフセットを使用して，異なったフレームアドレスを持つセグメント内の命令にアクセスしようとした場合，発生する可能性があります．このことは，また，グループ内のすべてのセグメントが単一の64Kバイトブロックのメモリ中に収まらなかった場合には，発生する可能性があります．

short 参照は，JMP 命令が，同一のセグメントまたはグループ内にあるラベルの付いた命令に制御を渡そうとするとき，行なわれます．目標命令は，参照点から 128 バイトのところになければなりません．LINK は，この参照のために，符号付きの 8 ビットの数を計算します．目標命令が異なったセグメントまたはグループに属する（異なったフレームアドレスを持つ）か，または目標までの距離が 128 バイトより遠い(いずれかの方向)場合，LINK はエラーメッセージを表示します．

near self-relative 参照は，命令が同一のセグメントまたはグループからの相対アドレスにあるデータにアクセスするとき，行なわれます．LINK は，この参照のために，16 ビットオフセットを計算します．このデータが同一のセグメントまたはグループ内にないとき，LINK はエラーを表示します．

near segment-relative 参照は，命令が，指定されたセグメントまたはグループ内のデータ，または指定されたセグメントレジスタからの相対アドレスにあるデータにアクセスしようとしたとき，行なわれます．LINK は，この参照のために，16 ビットのオフセットを計算します．指定されたフレーム内の目標のオフセットが 64K より大きいか，またはゼロより小さい場合，あるいはこの目標の標準フレームの開始点をアドレス指定できない場合，LINK はエラーメッセージを表示します．

long 参照は，CALL 命令が，他のセグメントまたはグループ内の命令にアクセスしようとしたとき行われます．LINK は，この参照のために，16 ビットフレームアドレスおよび 16 ビットオフセットを計算します．計算されたオフセットが 64 K より大きいか，またはゼロより小さい場合，あるいはこの目標の標準フレームの開始点がアドレス指定できない場合，LINK はエラーメッセージを表示します．

2.4.7 ローディングの順序の制御

実際のセグメントをロードしたい順序で，空のセグメント定義が示されているダミープログラムファイルを作成し，アセンブルすることによって，プログラム中のセグメントのローディングの順序を制御することができます．いったん，このファイルがアセンブルされると，LINK を呼び出すとき，単にこのファイルを先頭のオブジェクトファイルとして指定します．LINK は，これらのセグメントを指定された順序で自動的にロードします．

たとえば，つぎのダミープログラムファイルは，CODE，DATA，STACK，CONST，および MEMORY という名前のセグメントによって構成されるプログラム中のセグメントのローディングの順序を定義します．

```
CODE        segment     'CODE'
CODE        ends
CONST       segment     'CONST'
CONST       ends
DATA        segment     'DATA'
DATA        ends
STACK       segment     stack    'STACK'
STACK       ends
MEMORY      segment     'MEMORY'
MEMORY      ends
```

このダミープログラムファイルには，プログラム中で使用すべきすべてのクラスのための定義が入っていなければなりません．そうでない場合，LINKは省略時のローディングの順序を選択しますが，これはユーザーが希望する順序と一致する場合としない場合があります．プログラムをリンクするとき，このダミープログラムが，LINKコマンド行で指定された先頭のオブジェクトファイルでなければなりません．

注意

ダミープログラムファイルを，C, Pascal，または他の高級言語のプログラムと一緒に，ダミープログラムファイルと一緒に使用してはなりません．これらの言語は，それ自身のローディング順序を定義します．これらの順序を，修正してはなりません．

ダミープログラムファイルの終わりに空のMEMORYセグメントを置くことによって，MEMORYセグメントをプログラム中の最終セグメントとしてロードするようにLINKに強制することができます．空のセグメントは，つぎのように指定されなければなりません．

```
segment-name SEGMENT MEMORY 'class-name'
segment-name ENDS
```

segment-nameはMEMORYセグメントのために，class-nameはメモリクラスのために使用したい名前です．

例:

```
MEMORY      segment     memory      'MEMORY'
MEMORY      ends
```

2.5 メッセージ

リンカ；LINKの表示するメッセージをアルファベット順に解説します.

About to generate .EXE file.
Change diskette in drive <d:> and press <ENTER>

.EXEファイルを生成する処理を休止しています.EXEファイルを収めるディスクをドライブ<d:>にセット（交換）して，リターンキーを押してください．これは，／Pスイッチを指定したときに表示されるメッセージです．

Ambiguous switch error: 'x'

スイッチ名を省略しすぎて，一意に確定できません．スイッチの最小の省略形を確認してください．たとえば，つぎのようなスイッチの指定をすると，このメッセージが表示され，LINKは処理を中止します．

A > LINK ／N main;

Array element size mismatch

コモンなアレイ領域が，2ヶ所以上で，異なるサイズのエレメント（文字型と実数型など）で宣言されています．（現バージョンのMASMでは，コモンなアレイはサポートされていません．）

Attempt to put segment <name> in more than one group in file <filename>

<filename>中のセグメント<name>は，2つのグループで重複して定義されています．ソースファイルを修正して，オブジェクトファイルを作り直してください．

Bad value for cparMaxAlloc

／CPARMAXALLOCスイッチで指定した数値が， 1～65535の範囲にありませんでした．

Cannot find library: <filename>.lib. Enter new file spec:

指定されたライブラリファイル<filename>.libが見つかりません．ドライブ名，パス名，ファイル名を確認して，正しいファイル名を入力してください．

Cannot nest response files

応答ファイル中に，さらに応答ファイル名が含まれています．応答ファイルは入れ子にできません．応答ファイルを修正してください．

Cannot open list file

ディスクまたはディレクトリが一杯で，リストファイルを作成できません．ディスクまたはディレクトリの空きスペースを作ってください．

Cannot open response file

応答ファイルをオープンできません．ファイル名のミスタイプによる可能性が高いと考えられます．

Cannot open run file

ディスクまたはディレクトリが一杯で，出力（実行可能）ファイルを作成できません．ディスクまたはディレクトリの空きスペースを作ってください．

Cannot open temporary file

ディスクまたはディレクトリが一杯で，一時作業ファイル（VM.TMP）を作成できません．ディスクまたはディレクトリの空きスペースを作ってください．

Cannot reopen list file

LINK を起動したディスクが交換されたために，リストファイルを再オープンできませんでした．LINK を再起動してください．

Common area longer than 65536 bytes

ユーザープログラムが，64 K バイト以上のコモン変数を持っています．

Data record too large

オブジェクトモジュール中の LEDATA レコードに，1024 バイト以上のデータが入っています．

Dup record too large

オブジェクトモジュール中の LIDATA レコードに，512 バイト以上のデータが入っています．アセンブリモジュール中の構造定義が複雑すぎたり，DUP ステートメントのネストが深すぎる（例：ARRAY db 10 dup (11 dup (12 dup (13 dup (•••))))ことが原因と考えられます．ソースファイルを修正してください．

<filename> is not a valid library

指定したライブラリファイル <filename> に問題があります．

Fixup overflow near 〈num〉 in segment 〈name〉 in 〈filename〉(name) offset 〈num〉

このメッセージが出力される原因としては，つぎのような状況が考えられます．1) 1グループが64Kバイト以上ある．2) ユーザープログラム中に，セグメントにまたがるshortジャンプ／コールが含まれている．3) リンクするライブラリ中のルーチンと，同一のデータ項目名が使われている．4) セグメントの本体中で，EXTRNが用いられている．この例をつぎに示します．

```
CODE   segment public ' code'
extrn  main: far
start  proc    far
       call    main
       ret
start  endp
CODE   ends
```

前例を修正したものをつぎに示します．

```
extrn  main: far
CODE   segment public 'code'
start  proc    far
       call    main
               ret
start  endp
CODE   ends
```

ソースファイルを修正して，再度アセンブルしてください．

Incorrect DOS version, use DOS 2.0 or later

このLINKは，MS-DOS Ver 2.0以上でお使いください．

Insufficient stack space

メモリ不足で，LINKを実行できません．

Interrupt number exceeds 255

／OVERLAYINTERRUPTスイッチで，255よりも大きな値が指定されています．指定できる値の範囲は，4～255です．

Invalid numeric switch specification

スイッチに，無効なパラメータが指定されています．数値を指定するべきところで文字を指定している（0とOの間違い），なども原因として考えられます．

Invalid object module

オブジェクトモジュールに問題があります．アセンブルまたはコンパイルしなおしてください．

NEAR／HUGE conflict

コモン変数のnearとhuge定義に，重複が生じています．（現バージョンのMASMでは，コモン変数はサポートされていません．）

Nested left parentheses

オーバーレイを指定する，LINKのコマンドラインに，文法的な誤りがあります．

No object modules specified

オブジェクトファイルが指定されていません．

Out of space on list file

リストファイルを書き出し中のディスクが一杯になり，全部を収められません．ディスクの空きスペースをふやして，LINKを再起動してください．

Out of space on run file

EXE（実行可能）ファイルを書き出し中のディスクが一杯になり，全部を収められません．ディスクの空きスペースをふやして，LINKを再起動してください．

Out of space on scratch file

カレントドライブのディスクが一杯になってしまいました．ディスクの空きスペースをふやすなどして，LINKを再起動してください．

Overlay manager symbol already defined: <name>

ユーザーが定義したシンボル名と，オーバーレイマネージャ名が重複しています．名前を変更して，再処理してください．

Please replace original diskette in drive <d:> and press ENTER

LINKを起動した（オリジナル）ディスクをドライブ<d:>にセットして，リターンキーを押してください．これは，／Pスイッチを使用したときに表示されるメッセージで，EXEファイルを書き出し終わると出力されます．

Relocation table overflow

16384バイトを越す，longコール／ジャンプ／ポインタが使われています．long参照をshort参照に換えて，再処理してください．

Segment limit set too high

／SEGMENTスイッチで指定した数値が大きすぎます．

Segment limit too high

／SEGMENTスイッチで指定した値(または初期設定値：128)で，アロケートテーブルを作るには，メモリが不足しています．値を小さくするか，メモリのフリーエリアを広げてください．

Segment size exceeds 64 K

セグメントサイズの制限，64 Kバイトを越えています．たとえば，ユーザープログラムは，小規模メモリモデルでありながら64 Kバイト以上のコード，中規模メモリモデルでありながら64 Kバイト以上のデータを持っています．中規模・大規模メモリモデルで，コンパイルとリンク処理を行ってください．

Stack size exceeds 65536 bytes

／STACKスイッチで指定した値が，65536バイトを越えています．

Symbol table overflow

ユーザープログラムが，256 KB以上のシンボル情報（public, extrn, segment, group, class, fileなど）を含んでいます．セグメントやモジュールの結合などを行って，オブジェクトファイルを作り直してください．

Terminated by user

ユーザーの[CTRL]＋[C]キー入力によって，リンク処理が中止されました．

Too many external symbols in one modul

ユーザーのオブジェクトモジュール中の外部シンボルが多過ぎます．モジュールを分割するなどしてください．

Too many group-, segment-, and class-names in one module

ユーザープログラム中のグループ名，セグメント名，クラス名が多過ぎます．数を減らして，オブジェクトファイルを作り直してください．

Too many groups

ユーザープログラム中で，10個以上のグループを定義しています．

Too many GRPDEFs in one module

1モジュール中に, 10個以上のGRPDEFがあります. 数を減らすか, モジュールを分割してください.

Too many libraries

17個以上のライブラリファイルを指定しています. 16個以下にしてください.

Too many overlays

ユーザープログラム中で, 64個以上のオーバーレイを定義しています. 数を減らしてください.

Too many segments

ユーザープログラム中のセグメントが多すぎます. ／SEGMENTスイッチのパラメータを適当な数値にふやして, 再びリンク処理を行ってください.

Too many segments in one module

オブジェクトモジュールに, 256個以上のセグメントが含まれています. オブジェクトを分割するか, セグメントを連結して数を減らしてください.

Too many TYPDEFs

オブジェクトモジュール中のTYPDEFレコードが多過ぎます. このレコードは, コンパイラがコモン変数の領域のために生成したものです. (現バージョンのMASMは, コモン変数をサポートしていません.)

Unexpected end-of-file on library

ライブラリを収めたディスクが, 外された可能性が高いと考えられます. ディスクを確認してください.

Unexpected end-of-file on scratch file

VM.TMPファイルの作られていたディスクが, 外された可能性が高いと考えられます. ディスクを確認して, LINKを再起動してください.

Unmatched left parenthesis

オーバーレイを指定するLINKのコマンドラインに, 文法エラーがあります.

Unmatched right parenthesis

オーバーレイを指定するLINKのコマンドラインに, 文法エラーがあります.

Unrecognized switch error: '<filename>'

無効な文字が，スイッチとして指定されています．たとえば，つぎのようなスイッチは存在しません．

```
A> LINK /ABCDEF main;
```

LINKは処理を中止します．

VM.TMP is an illegal file name and has been ignored

ユーザーのオブジェクトファイル名に“VM.TMP”が使われています．ファイル名を変更してください．

Warning: no stack segment

ユーザープログラムは，スタックスペースセグメントを割り当てるステートメントを含んでいません．

Warning: too many local symbols

ローカルシンボルのソート（並び換え）リストを要求しましたが，シンボル数が多過ぎて，ソートできませんでした．LINKは，ソートしていないローカルシンボルのリストを作成しました．

Warning: too many public symbols

パブリックシンボルのソート（並び換え）リストを要求しましたが，シンボル数が多過ぎて，ソートできませんでした．LINKは，ソートしていないパブリックシンボルのリストを作成しました．

第3章
SYMDEB：シンボリックデバッグユーティリティ

3.1 イントロダクション

SYMDEBは，管理された検査用の環境を提供するデバッガで，実行可能なプログラムやその他のバイナリファイルのロード，検査や修正を行います．

SYMDEBでは，プログラムコードの表示，レジスタの検査，ブレークポイントのセットおよび命令実行のトレースが可能であり，マイクロソフト言語が使用する浮動小数点エミュレーションの慣例を用いたプログラムをデバッグできます．

SYMDEBは，シンボリックデバッグユーティリティであり，データや命令の参照をアドレスではなく，名前（シンボル）で行うことができます．名前を入力すれば変数の表示やルーチンの実行ができ，アドレスを入力する必要はありません．

SYMDEBでは，プログラム内のパブリックシンボルのアドレス情報の入ったシンボルマップファイル（.SYMファイル）をいれることができます．シンボルマップファイルは，MAPSYM（シンボルマップファイルユーティリティ）およびLINKの作るマップファイルを用いて作成します．

SYMDEBのブレークポイントコマンドは，プログラムコードのブレークポイントのセット，有効化，表示およびクリアを実行します．このブレークポイントは，どんな命令にあっても，プログラムの実行を停止させ，無効化するか削除するまでは有効です．

SYMDEBの式評価機能は，シンボルと数に関する演算を行います．セグメント値やオフセットシンボルからの抽出を含むすべての算術演算，ブール演算およびアドレスの演算が可能です．オペランドの参照は，値，アドレスまたはポート番号で行います．

●ソース行の表示と番号付け

SYMDEBの強力な機能として，ソース行の表示と番号付けがあります．この機能によって，CやPascal, FORTRANなどのプログラムのデバッグが，機械レベルだけでなく，ソースファイルレベルでも行えます．

表示レベルはSYMDEBコマンドで選択でき，プログラムの実際のソース文や機械語，またはそれらを組み合わせたものの表示が可能です．

さらにSYMDEBは，表示コマンドとブレークポイントコマンドへの引数として，ソース行番号を受け取ります．これによって，ソースファイルの個々の文の参照は，そのメモリ内の正確な位置を知らなくても行うことができます．

本章では，SYMDEB の使用方法，特にプログラムをデバッグする際のデバッガの起動方法，シンボル（.SYM）ファイルの作成方法および使用方，SYMDEB コマンドの使用方法について説明します．

3.2　SYMDEB の起動

SYMDEB は，MS-DOS のコマンドプロンプトにその名前をタイプすれば起動できます．SYMDEB を呼び出すときの書式をつぎに示します．

SYMDEB ［〈シンボルファイル名〉］［〈デバッグファイル名〉［〈引数リスト〉］］

シンボルファイル　シンボルファイル（.SYM）に名前をつけるオプションのファイル名．

ファイル名　ロードするバイナリファイルや実行可能ファイルに名前を付けるオプションのファイル名．

引数リスト　オプションのプログラム引数のリスト．SYMDEB では，ロードされたプログラムがこの引数リストを利用できるようになっている．

起動後，SYMDEB はメッセージを表示し，つづいて SYMDEB プロンプト（-）を表示します．プロンプトが示されたら，SYMDEB コマンドを入力できます．

例：

```
A > symdeb
Microsoft Symbolic Debug Utility
Version 3.01
(C)Copyright Microsoft Corp 1984, 1985
Processor is  [80286]
_
```

前記の例において，"symdeb"は，MS-DOS コマンド行にタイプされます．コマンドは，始動メッセージと SYMDEB プロンプト（-）を表示します．

3.2.1　デバッグファイルを指定したSYMDEBの起動

SYMDEBを起動するコマンドライン上に，デバッグしたいファイル名を指定することによって，実行可能なプログラムファイル（.EXEまたは.COM）をロードできます．たとえば，SYMDEBの起動時にファイル〝file.exe″をロードする場合には，つぎのようにタイプします．

```
symdeb  file.exe
```

プログラムファイルがロードされた場合，SYMDEBは，使用可能なメモリの最下位のセグメントに，256バイトのプログラム用のヘッダを作成し，次にファイルの内容をヘッダの直後の空きメモリにコピーします．SYMDEBは，ファイルの大きさ（バイト単位）をBX:CXレジスタにコピーし，セグメントと他のレジスタをファイルが定義する個々の値に合わせます．

3.2.2　シンボルファイルを用いたSYMDEBの起動

シンボル演算を行う場合は，コマンド行に1つまたは複数のシンボルファイルを入力してSYMDEBを起動します．シンボルファイルを入力すると，SYMDEBは，入力されたファイルをロードし，そのファイルが定義するシンボルの使用を可能にします．

ロードするシンボルファイルは，MAPSYMコマンドで作成します．後述の〝シンボルファイルの作成″の項を参照してください．

たとえば，プログラムファイル〝file.exe″と共に，シンボルファイル〝file.sym″をロードする場合の入力は，つぎのようになります．

```
symdeb  file.sym  file.exe
```

SYMDEBは，〝file.sym″からメモリへとシンボル情報をコピーし，プログラムのヘッダを作成した後に，〝file.exe″をロードします．

複数のシンボルファイルを入力することもできます．一般に，複数のシンボルファイルは，別々にリンクされた数個のプログラムファイルからなるプログラムに使用します．しかし，すべてのシンボルファイルは，必ずプログラムファイル名の前に入力しなければなりません．プログラムファイルの後に入力されたファイルは，すべてプログラム引数とみなされます．

シンボルをロードする場合には，プログラムファイルをロードする必要はありません．SYMDEBは，ファイル名を用いずに起動したかのように起動します（〝ファイルを用いないSYMDEBの起動″の項参照）．

3.2.3　引数のロード済プログラムへの引き渡し

SYMDEBコマンドライン上のプログラムファイル名の直後に引数をタイプすることによって，ロード済のプログラムへ，1つまたはそれ以上のプログラム引数を引き渡すことができます．SYMDEBは，すべての引数を入力されたとおりに，プログラムのヘッダにコピーします．

たとえば，データファイル〝test.dat〟と，オプションの〝／m〟と〝／b〟をプログラムファイル〝file.exe〟へ移動させる場合は，つぎのようにタイプします．

```
symdeb  file.sym  file.exe  test.dat  /m  /b
```

SYMDEBは文字列〝test.dat ／m ／b〟をプログラムヘッダに置き，それから〝file.exe〟をロードします．引数をロードしてしまえば，〝file.exe〟はプログラムからいつでも引数を読み取ることができます．

プログラムヘッダの詳細については，後述の〝Nameコマンド〟の項を参照してください．

3.2.4　ファイルを用いないSYMDEBの起動

SYMDEBは，ファイルを用いなくても，〝SYMDEB〟とタイプするだけで起動できます．ファイル名を用いないでSYMDEBを起動すると，SYMDEBはプログラムのヘッダを作成しますが，プログラムのロードは行いません．この場合，NameコマンドとLoadコマンドを自由に用いて，希望のファイルに名前を付けたり，それをロードしたりできます．

ファイルを用いないでSYMDEBを起動すると，SYMDEBは空きメモリの最下位にセグメントレジスタをセットし，命令ポインタを0100Hにセットし，すべてのフラグをクリアし，他のレジスタを0にセットします．

3.2.5　シンボルファイルの作成

SYMDEBがデバッグするプログラムで使用するシンボルファイルは，MAPSYMプログラムを用いて作成します．このプログラムは，プログラムのシンボルマップファイル（.MAP）の内容の書式をSYMDEBを用いたロードに合うように変換します．

MAPSYMコマンドの書式は，つぎのとおりです．

```
MAPSYM  [－l|／l] ファイル名
```

ファイル名は，リンク時に作成されるシンボルやマップ(.MAP)のファイルのことで，ファイル名の拡張子を入力しないと〝.MAP〟と解釈されます．シンボルマップファイルは，マップファイルを指定して，LINKコマンドでスイッチ〝／MAP〟を指定すれば作成できます．ソース行のデバッグを行いたい場合は，同様に，スイッチ〝／LINENUMBER〟を入力する必要があります．詳しくは，第2章〝LINK：リンカ〟を参照してください．

任意選択の"−l"と"／l"は同じもので，プログラムに定義されたグループ名，プログラムの開始および行番号の有無などの変換に関する情報を，MAPSYMに表示させます．

たとえば，マップファイル"file.map"からシンボルファイル"file.sym"を作成する場合は，つぎのようにタイプします．

```
mapsym  file.map
```

MAPSYMプログラムは新しいファイルを作成し，シンボル情報をそのファイルにコピーします．

注意

作成するシンボルファイルのファイル名としてパスを指定することはできません．

3.3　コントロールキーの使用

誤りの修正やコマンドの停止は，MS-DOSユーザーズガイドで述べられているコントロールキャラクタやテンプレート機能を用いれば行えます．以降の項では，コマンドの停止方法と保留方法について述べます．

3.3.1　SYMDEBコマンドの停止

SYMDEBコマンドは，[CTRL]+[C]キーを押せば，いつでも停止させることができます．このキーによってSYMDEBは，実行中のコマンドを停止して，SYMDEBプロンプトを表示します．

一般に，[CTRL]+[C]キーは，長いDumpコマンドを停止する場合に用います．このキーは，無限ループを入力したGoコマンドから制御を取り除く場合にも使用できますが，それは，プログラムが入力または出力処理を実行している場合に限られます．

注意

SYMDEBの入力がRedirectionコマンドを用いて"AUX"に向けられた場合は，[CTRL]+[C]キーは無効で，無視されます．

3.3.2 コマンドの保留

SYMDEBの出力は，CTRL+Sキーを押せば，一時的に保留させることができます．SYMDEBは，別のキーを押されるまでは，コマンドの出力を中断します．一般に，CTRL+Sキーは，特定のバイトや命令を検査するために，DumpコマンドやUnassembleコマンドの出力を一時的に止める場合に用います．

注意 SYMDEBの入力がRedirectionコマンドを用いて〝AUX〞に向けられた場合は，CTRL+Sキーは無効で，無視されます．

3.4 コマンド

次の表に，すべてのSYMDEBコマンドを示します．

?	値の表示，ヘルプ表示	EW	ワード入力
<	入力の切り換え	ED	ダブルワード入力
>	出力の切り換え	ES	短い実数の入力
=	入力と出力の切り換え	EL	長い実数の入力
!	MS-DOSココマンドの実行	ET	10バイト実数の入力
.	現在のソースラインの表示	F	フィル
*	コメントの表示	G	実行
A	アセンブル	H	16進計算
BP	ブレークポイントの作成	I	ポート入力
BC	ブレークポイントのクリア	K	スタックフレームの表示
BD	ブレークポイントの無効化	L	ロード
BE	ブレークポイントの有効化	M	移動
BL	ブレークポイントのリスト	N	名前のセット
C	比較	O	ポート出力
D	ダンプ	P	Pトレース
DA	ASCIIダンプ	Q	中止
DB	バイトダンプ	R	レジスタ
DW	ワードダンプ	S	サーチ，ソース形式のセット
DD	ダブルワードダンプ	T	トレース
DS	短い実数のダンプ	U	逆アセンブル
DL	長い実数のダンプ	V	ソースラインの表示
DT	10バイト実数のダンプ	W	書き込み
E	入力	X	シンボルマップの検査
EA	ASCIIコード値入力	XO	シンボルマップの解放
EB	バイト入力	Z	シンボリックアドレスに値をセット

3.4.1 コマンド書式

すべてのSYMDEBコマンドの一般書式はつぎのとおりです．

〈コマンド名〉 〈パラメータ〉…

〈コマンド名〉は，1・2文字のアルファベットによるコマンド名です．

〈パラメータ〉は，コマンドが使用する値やアドレスを表す数やシンボルや式です．コマンド名とパラメータには，大文字と小文字をどのように組み合わせて用いても構いません．

コマンドのパラメータの数は，そのコマンドによって異なります．コマンドが複数のパラメータを持つ場合は，1つのカンマ（,）またはスペース（空白）によってパラメータを区切りながら並べます．

例：

```
D   cs:100  110
U   cs:100  110
F   ds:100, 110  ff, fe, 01, 00
```

以後の項では，コマンドのパラメータについて詳しく説明します．

3.4.2 シンボル

シンボルは，レジスタ，絶対値，セグメントアドレス，またはセグメントのオフセットを表す名前です．シンボルは，1つまたは複数の文字から構成されますが，必ず，文字，アンダースコア（_），疑問符（?），アット記号（@），ドル記号（$）で始まります．

レジスタを指名するシンボルは，常に使用可能です(すべての表については，〝Registarコマンド″の項を参照してください)．他のシンボルは，そのシンボルの名前や値を定義するシンボルマップファイルがロードされている場合にのみ使用することができます．

注意

SYMDEBは，大文字と小文字を，同じ文字として扱います．

スペルの違いが大文字・小文字だけであるシンボルは，同じものとして扱われます．シンボルマップファイルにそのような2つのシンボルがある場合，SYMDEBはそのうちの1つだけを承認します．もう一方のシンボルについての情報をアクセスしようとすると，常に最初のシンボルについての情報が返ってきます．

レジスタ名と同じスペルのシンボルは無視されます．レジスタ名は，常にそういったシンボルに優先します．

例：

AX

main

DGROUP

IP

3.4.3　数

SYMDEBでは，2，8，10，16進数を用いることができ，添え字で基数を表します．数字の表記法の書式はつぎのようなものです．

〈数字〉 Y

〈数字〉 O

〈数字〉 Q

〈数字〉 T

〈数字〉 H

数は整数を表し，2進数字，8進数字，10進数字，または16進数字とオプションの基数を組み合わせたものです．数字は，Y，O，Q，T，Hのうちから指定された基数の数字です．基数が入力されない場合は，H（16進）と解釈されます．

つぎの表で，各々の基数で使用できる数字を示します．

添え字	タイプ	数字
Y	2進数	01
O，Q	8進数	01234567
T	10進数	0123456789
H	16進数	0123456789 ABCDEF

数には，数字をいくつでもいれることができますが，SYMDEBでは，数が65535より大きい場合は，先頭の数字を切ります．また，先頭にゼロがあっても無視されます．

例：

```
0111111Y        77Q      63T    3FH    3F
01001010100101Y   112450   4773T  12AH   12A5
```

3.4.4 アドレス

アドレスの指定の書式は，つぎのようなものです．

〈セグメント〉：〈オフセット〉

アドレスは，セグメントのアドレスを表す値と，セグメント中のオフセットを表す値の，2つの16ビットの値を組み合わせたものです．組み合わせた値が，特定のメモリアドレスを指定します．

完全なアドレスは，セグメントアドレスと，オフセットを持っており，この2つはコロン(：)で区切られます．部分アドレスは，オフセットのみです．どちらの場合も，セグメントやオフセットは，数かレジスタ名またはシンボルです．ほとんどのコマンドの場合，部分アドレスには，省略時セグメントアドレスが与えられます．省略値は，DSセグメントレジスタの最新の内容です．Assembleコマンド，Goコマンド，Loadコマンド，PTraceコマンド，Unassembleコマンド，およびWriteコマンドの場合，省略時のセグメントアドレスは，CSレジスタの内容です．

例：

```
cs : 0100
04BA : IP
CS : main
DGROUP : count
```

3.4.5 アドレス範囲

アドレス範囲の書式は，つぎのようなものです．

〈開始アドレス〉　〈終了アドレス〉

範囲は，連続している記憶位置の順序を指定する2つの記憶アドレスを組み合わせたものです．開始アドレスと終了アドレスは，範囲の最初と最後のアドレスを指定します．

コマンドがレンジをとった場合に，2番目のアドレスを入力しないと，SYMDEBは常に128バイトの範囲を想定します．コマンドがレンジを取ってからすぐに3番目のパラメータを続けた場合は，2番目のアドレスを入力しなければなりません．そうしないと，SYMDEBは，3番目のパラメータを2番目のアドレスとして使用しています．

例：

```
cs : 100  110
__main  __main + 20
```

3.4.6　対象レンジ

対象レンジの書式は，つぎのようなものです．

〈開始アドレス〉　L　〈値〉

対象レンジは，メモリアドレスと，メモリ内の連続したバイト，ワード，命令，またはその他の対象の範囲を指定する〝対象物〞の総計を組み合わせたものです．開始アドレスは，リストの最初の対象のアドレスを指定し，〝L　値〞は，リスト内の対象数を指定します．

対象レンジは，Dump コマンド，Fill コマンド，Search コマンド，Unassemble コマンドでのみ用いることができます．各コマンドはリスト内の対象の大きさや種類を決定します．たとえば，Dump　Bytes コマンドはバイトを持ち，Dump　Words コマンドはワードを，Unassemble コマンドは命令を持っています．

例：

```
DGROUP : table  L  100
main  L  20
```

3.4.7　行番号

行番号は，ソースプログラムファイル中のテキストを含む行を示します．行番号の書式は，つぎのようなものです．

.±行番号
.［ファイル名：］番号
.シンボル［±行番号］

行番号は，10進数とファイル名とシンボルを組み合わせたもので，プログラムソースファイル内のテキストの特定の行を指定します．

最初の形式の書式は，相対行番号を指定するものです．番号は，現在のソース行から新しい行までのオフセット（行単位）です．プラス記号（＋）を入力した場合，新しい行は，ソースファイルの終わり寄りになります．マイナス記号（－）を入力した場合には，新しい行は，ファイルの先頭寄りになります．現在の行番号がない場合や指定した行番号に対するソース行がない場合，SYMDEB はエラーを表示します．

2番目の形式の書式は，絶対行番号を指定するものです．ファイル名を入力すると，指定された行が，ファイル名によって示されたシンボルファイルに対応するソースファイル内にあると解釈されます．ファイル名を入力しないと，現在の命令アドレス（たとえば，CS レジスタや IP レジスタの現在の値）が，その行を含んでいるソースファイルを決定します．ファイル名がない場合や，指定した行に対するソース行がない場合，SYMDEB はエラーの表示を行います．

3番目の形式の組み合わせは，シンボル行番号を指定します．シンボルは，どんな命令，また

は手続きラベルであっても構いません．番号を入力すると，その数は，特定ラベルや手続き名から新しい行までのオフセット（行単位）となります．プラス記号（+）を入力した場合，新しい行は，ソースファイルの終わり寄りになります．マイナス記号（−）を入力した場合には，新しい行は，ファイルの先頭寄りになります．シンボルがない場合や，指定した行に対するソース行がない場合，SYMDEB はエラーの表示を行います．

例：

表記法	意　味
. +5	;現ラインから 5 行下の行
. 10	;現ソースファイルの 10 行目
. sample : 1	; 〝sample〟が指定するソースファイルの 10 行目
. main	;ルーチン〝main〟の 1 行目
. main+5	;ルーチン〝main〟の 5 行目

シンボル〝main〟は，行番号の指定にも使用できます．この場合，〝main〟は〝. main〟と同じです．ただし，〝main+3〟が〝main〟から 3 バイト目のアドレスを指定するのに対して，〝main+3〟は，〝main〟から 3 行目のソース行を指定するので注意してください．

3.4.8 ストリング

ストリングは，引用符で囲まれた文字列で，SYMDEB は文字の ASCII 値を用います．ストリングの書式は，つぎのようなものです．

'文字'
〝文字〟

ストリングは，ASCII 値の<リスト>を表します．ストリングはどのような長さでもよいく，シングルコーテーション（'）かダブルコーテーション（〃）を含めることもできます．ただし，始まりの引用符と終わりの引用符は同じものでなくてはなりません．ストリング中にさらに引用符を入れるときは，SYMDEB がそのストリングを早目に終わらせるのを防ぐために，その引用符を 2 個つづけて入力するか，別の種類の引用符を用います．

例：

```
'This is a string.'
"This is a string."
'This "string" is okay.'
"This" "string" " is okay."
'This "string" is okay.'
"This 'string' is okay."
```

3.4.9　式

式は，8ビット・16ビット・32ビットの値を求めるパラメータと演算子の組み合わせです．式は，すべてのコマンドで，値として用いることができます．

式は，すべてのシンボルと数字，アドレスを，つぎに示す単項演算子および2項演算子を用いて組み合わせることができます．

●単項演算子

演算子	意　味	優先権
+	単項プラス	高
−	単項マイナス	↑
NOT	1の補数	
SEG	オペランドのセグメントアドレス	
OFF	オペランドのアドレスオフセット	
BY	特定アドレスからの下位バイト	
WO	特定アドレスからの下位ワード	
DW	特定アドレスからのダブルワード	
POI	特定アドレスからのポインタ（4バイト）	
PORT	特定ポートからの1バイト	↓
WPORT	特定ポートからのワード	低

● 2項演算子

演算子	意　味	優先権
*	乗算	高
／	整数の除算	↑
MOD	余り	
:	セグメントのオーバーライド	
+	加算	
−	減算	
AND	論理積	
XOR	排他的論理和	↓
OR	論理和	低

式は，演算子の優先順に実行されます．隣りあった演算子の優先順位が同じ場合，式は左から右へ評価されます．カッコを用いれば，この順序を変えることができます．

例：

式	式の値
4+2 * 3	;=10 (0AH)
SEG 0001:0002	;=1
OFF 0001:0002	;=2
4+(2 * 3)	;=10 (0AH)
(4+2) * 3	;=18 (12H)

3.5 SYMDEB のコマンド

本節では，SYMDEB のコマンドをアルファベット順に解説します．

SYMDEB のコマンドは，1·2 文字のアルファベットとそれにつづくいくつかのパラメータの形をしています．パラメータには省略することができるものがあり，そのときは，SYMDEB の初期設定値や直前に行ったコマンドの結果のレジスタ値が用いられます．

Assemble

書　式　A [<アドレス>]

機　能　8086／8087／8088／80186／80286／80287のニーモニックをアセンブルしてメモリへ直接入れる

解　説　Assembleコマンドは，8086／8087／8088／186／286／287ニーモニックをアセンブルして，その結果の命令コードを，メモリの指定されたアドレスに入れます．（286のプロテクトモードは除く）コマンドは，アドレスと指定されたアドレスの命令コードを表示して，次の命令のためのプロンプトを出します．アドレスを入力しないと，CSレジスタとIPレジスタの現在値によって与えられるアドレスからアセンブルが開始されます．

命令は，標準のニーモニックで入力できます．新しい命令をアセンブルする場合は，適切なニーモニックを入力してからリターンキーを押します．SYMDEBは，命令をアセンブルして，次に使用可能なアセンブルを表示します．アセンブルを終了させて，SYMDEBプロンプトに戻す場合は，リターンキーのみを押します．

命令ニーモニックの入力に関する規則を次に示します．

1. WAITやREPなどのプレフィクスニーモニックは，それらを適用する命令の前に指定します．これらは，別の行に分けて入力します．
2. FARリターンのためのニーモニックは，RETFです．
3. ストリング（文字列）の操作に関するニーモニックは，ストリングサイズを明確に指定します．たとえば，ワード列を移動する場合にはMOVSWを用い，バイト列を移動する場合は，MOVSBを使用します．
4. SYMDEBは，short，near，farの飛び越し（ジャンプ）や呼び出し（コール）を自動的に正しいアドレスにアセンブルします．これらは，NEARプレフィクスかFARプレフィクスを用いれば無効にすることができます．次に例をしめします．

```
JMP     502
JMP     NEAR    505
JMP     FAR     50A
```

NEARプレフィクスは，NEと略すことができます．ただし，FARプレフィクスを略すことはできません．

5. SYMDEB では，ワードの記憶位置を表すのか，バイトの記憶位置を表すのかわからないオペランドがあります．この場合，プレフィクス "WORD PTR" か "BYTE PTR" を用いて，データの種類を明確にしなくてはなりません．省略語として，"WO" と "BY" とが使用できます．つぎに例を示します．

```
MOV    WORD   PTR    [BP], 1
MOV    BYTE   PTR    [SI-1], SYMBOL
```

6. オペランドが記憶位置を示すのか，そのまま値オペランドを示すのか直接判断できないものがあります．SYMDEB では，角形カッコで囲まれたオペランドは，メモリを示すという慣例を用いています．つぎに例を示します．

```
MOV    AX, 21
MOV    AX, [21]
```

7. DB オペコードは，バイト値をメモリに直接アセンブルします．DW オペコードは，ワード値をメモリに直接アセンブルします．つぎに例を示します．

```
DB     1, 2, 3, 4, "THIS IS AN EXAMPLE"
DB     'THIS IS A QUOTE:"'
DB     "THIS IS A QUOTE:'"
DW     1000, 2000, 3000, "BACH"
```

8. SYMDEB は，すべての形式のレジスタ間接コマンドをサポートしています．つぎに例を示します．

```
ADD    BX, 34 [BP+2] [SI-1]
POP    [BP+DI]
PUSH   [SI]
```

9. すべての同義オペコードも同様にサポートしています．つぎに例を示します．

```
LOOPZ  100
LOOPE  100
JA     200
JNBE   200
```

10. 8087オペコードは,明確に指定しなくてはいけません.たとえば,WAIT は8086オペコード,FWAITは8087オペコードです.また,8087オペコードを使用するときは,システムに8087数値演算プロセッサが着装されている必要があります.つぎに例を示します.

```
FWAIT
FADD        ST, ST (3)
```

入力に誤りがある場合,SYMDEBは,"Error"というメッセージを表示し,現在のアセンブルアドレスを再び表示します.

例　Assembleコマンドの使用例をつぎに示します.

```
A  CS:__main
```

この例では,"CS:__main"によって与えられるアドレスからアセンブルが開始されます.

```
A  04BA:0100
```

この例では,"04BA:0100"によって与えられるアドレスからアセンブルが開始されます.

BreakPoint set

書 式　BP [n] 〈アドレス〉 [〈回数〉]

機 能　ブレークポイントの作成

解 説　Breakpoint Set コマンドは，入力されたアドレスに〝強力な〟ブレークポイントを作成します．プログラム実行時にブレークポイントに達すると，プログラムは停止して，SYMDEB はすべてのレジスタとフラグの現在値を表示します．強力なブレークポイントは，Go コマンドが作成するブレークポイントと違って，Breakpoint Clear コマンドを用いて一時的に無効にしない限り，プログラム内に留まります．

SYMDEB では，強力なブレークポイントを 10 個まで用いることができます(0 から 9)．[n] は，作成するブレークポイント番号の指定です．BP と n の間にスペース(空白)を入れてはいけません．n が入力されない場合は，最初に使用可能なブレークポイント番号が使用されます．〈アドレス〉は，有効な命令のアドレス（つまり命令オペコードの最初のバイト）です．〈回数〉は，ブレークポイントが採用されるまでに無視される回数を指定します．

例　BP コマンドの使用例をつぎに示します．

```
BP__main
```

この例では，強力なブレークポイントが〝__main〟に作成されています．

```
BP8 add
```

この例では，ブレークポイント 8 番がアドレス〝add〟に作成されます．

```
BP  100  10
```

この例では，ブレークポイントは，現 CS セグメントのアドレス 100 に作成されます．このブレークポイントは，採用されるまでに 16（10H）回無視されます．

Breakpoint Clear

書式　BC　{<リスト>　|　*}

機能　ブレークポイントの削除

解説　Breakpoint Clear コマンドは，強力なブレークポイントをプログラムから取り除きます．<リスト>を入力すると，そこに挙げられているブレークポイントを複数削除します．リストは，0から9の整数値の組み合わせです．アスタリスク（*）を指定すると，すべてのブレークポイントを取り除きます．

例　BCコマンドの使用例をつぎに示します．

```
BC 0 4 8
```

この例では，ブレークポイント0，4，8番が取り除かれます．

```
BC *
```

この例では，すべてのブレークポイントが取り除かれます．

Breakpoint Disable

書式　BD {<リスト> | *}

機能　ブレークポイントの一時的な無効化

解説　Breakpoint Disable コマンドは，強力なブレークポイントを無効にします．ブレークポイントは削除されるわけではなく，Breakpoint Enable コマンドを用いて，いつでも復元できます．

<リスト>を入力すると，コマンドは，そこに挙げられている番号のブレークポイントを無効にします．<リスト>の内容は，0 から 9 の整数値の組み合わせです．アスタリスク（*）を入力すると，すべてのブレークポイントを無効にします．

例　BD コマンドの使用例をつぎに示します．

```
BD 0 4 8
```

この例では，ブレークポイント 0，4，8 番が無効になります．

```
BD *
```

この例では，すべてのブレークポイントが無効になります．

Breakpoint Enable

書　式　BE　{<リスト> | *}

機　能　ブレークポイントの有効化

解　説　Breakpoint Enableコマンドは，Breakpoint Disableコマンドによって一時的に無効となった強力なブレークポイントを有効化（復元）します．

<リスト>を入力すると，そこに挙げられている複数のブレークポイントの有効化を行います．リストは，0から9の整数値の組み合わせです．アスアリスク(*)を入力すると，すべてのブレークポイントが有効化されます．

例　BEコマンドの使用例をつぎに示します．

```
BE 0 4 8
```

この例では，ブレークポイント0，4，8番が有効化されます．

```
BE *
```

この例では，すべてのブレークポイントが有効化されます．

Breakpoint List

書　式　BL

機　能　ブレークポイントのリスト

解　説　Breakpoint List コマンドは，Breakpoint コマンドが作成するすべてのブレークポイントに関する現在の情報をリストアップします．コマンドは，ブレークポイント番号，有効化／無効化の状態，ブレークポイントのアドレス，〈回数〉の残り数，〈回数〉の設定値（カッコ内）を表示します．

ブレークポイントが設定されていなければ，リストは何も表示されません．

例　BL コマンドの使用例をつぎに示します．

```
BL
```

このように入力すると，ブレークポイントの状況が表示されます．表示はつぎのようになります．

```
0  e  04BA:0100
4  d  04BA:0503  4 (10)
8  e  0D2D:0001  3 (3)
```

この例では，ブレークポイント 0 と 8 は有効化 (e) されていて，ブレークポイント 4 は無効化 (d) されています．ブレークポイント 4 の〈回数〉の設定は 10 で，あと 4 回の残り回数があります．ブレークポイント 8 の最初のパス数は 3 で，その 3 つの〈回数〉をすべて残しています．ブレークポイント 0 には，最初の〈回数〉がありませんが，これは 1 にセットされたということです．

Comment

書　式　＊　〈コメント〉

機　能　注釈文（コメント）の表示

解　説　Comment コマンドは，アスタリスク（＊）につづいてタイプした文字をスクリーンなどの出力装置にエコーバックします．このコマンドは，Redirection（入出力の切り替え）コマンドと組み合わせて用いると便利で，デバッグ作業の流れ（経過・工程）を記録として残しておきたいときなどに役立ちます．

例　Comment コマンドの使用例をつぎに示します．

```
-R  CX  80
-*  Change  the  count  in  CX  to  80
Change  the  count  in  CX  to  80
```

Compare

書　式　C 〈レンジ〉〈アドレス〉

機　能　メモリ内容の比較

解　説

Compare コマンドは，〈レンジ〉が指定する範囲のメモリのバイトと，〈アドレス〉から始まるそれに対応するメモリのバイトを比較します．対応するバイトがすべて一致したときは，SYMDEB はプロンプトを表示して，次のコマンドを待ちます．対応するバイトが一致しない場合は，その不一致のバイトが表示されます．

例

C コマンドの使用例をつぎに示します．

```
C  100, 1FF    300
```

この例では，メモリの 100H 番地から 1FFH 番地までのブロックが，メモリの 300H 番地から 3FFH 番地までのブロックと比較されます．

```
C  100  L  100    300
```

この例では，メモリの 100H 番地から 256 (100H) バイト目までが，300H 番地から 256 バイト目までと比較されます．

Display

書 式　? 〈式〉

機 能　〈式〉の値の表示

解 説　Displayコマンドは，入力された〈式〉の値を表示します．コマンドは，式を評価してから，様々な形式でその値を表示します．形式には，完全アドレス16ビット16進数，32ビット16進数，10進数（カッコで囲まれている），ストリング値（ダブルコーテーションで囲まれている）があります．

〈式〉は，数，シンボル，アドレスおよび演算子を組み合わせたものです．演算子については，この章で述べた"式"の項を参照してください．

例　?（Display）コマンドの使用例をつぎに示します．

```
? 3 * 4
```

この例では，式"3 * 4"の値が表示されます．

```
? DS:table
```

この例では，シンボルアドレス"DS:table"の値が表示されます．

```
? WO DGROUP:__bufsiz
```

この例では，シンボルアドレス"DGROUP:__bufsiz"のワードが表示されます．

Dump Ascii

書 式 DA [〈アドレス〉|〈レンジ〉]

機 能 メモリ内容の ASCII ダンプ

解 説 Dump Ascii コマンドは，指定された〈アドレス〉または<レンジ>のメモリの内容を ASCII 文字で表示します．コマンドは，入力するアドレスまたはレンジによって，1行または複数行の文字を表示します．キャリッジリターンやラインフィードなどの印字できないコードは，ピリオド（.）で表示されます．

〈アドレス〉を入力すると，最初の〝0″バイトにあたるまで，または 128 バイトを表示しきるまで，ASCII 文字を表示し続けます．

〈レンジ〉を入力すると，コマンドは，その範囲が終わるまで，ASCII 文字の表示を続けます．アドレスもレンジも入力されない場合，コマンドは最初の〝0″バイトまでのすべての文字するか，または 128 バイトを表示してしまうまで，ASCII 文字の表示を行います．この表示は，前回に表示された最終バイトの直後のアドレスである現ダンプアドレスから始まります．

例 DA コマンドの使用例をつぎに示します．

```
DA  CS:100  100
```

この例では，〝CS:100″から″CS:110″までのバイトの ASCII 値が表示されます．表示はつぎのようになります．

```
04BA:0100  A STRING.. Text..
```

このように，印字できないコードはピリオド（.）で表示されます．

```
DA
```

この例では，現ダンプアドレスの文字が表示されます．たとえば，前回の Dump Ascii コマンドでの最終バイトが 04BA:0110 であった場合，このコマンドでは，04BA:0111 からバイトの表示が開始されます．

```
DA  name
```

この例では，シンボルアドレス〝name″の文字が表示されます．

Dump Bytes

書　式　DB　[〈アドレス〉|〈レンジ〉]

機　能　**メモリ内容のバイト単位の16進と，ASCIIダンプ**

解　説　Dump Byteコマンドは，入力された〈アドレス〉または〈レンジ〉のメモリ内容を16進値とASCII値で表示します．入力する〈アドレス〉または〈レンジ〉によって，コマンドは，1行または複数行の表示を行います．　各行には，その行の先頭バイトのアドレスが表示され，それに続いて，最大16個の16進バイト値が表示されます．バイト値の直後に，対応するASCII値が続きます．16進値は，ハイフン(-)で区切られる8番目の値と9番目の値以外は，スペースでピリオドられます．ASCII値は，区切らないで印字されます．印字できないASCII値は，( )で示されます．コマンドは，レンジが終わるまで，または最初の128バイトが表示されるまで，値と文字の表示を行います．

例　DBコマンドの使用例をつぎに示します．

```
DB  CS:100  110
```

この例では，"CS:100" から "110" までのバイト値が表示されます．表示は，つぎのようになります．

```
04BA:0100 41 20 73 74 72 69 6E 67-04 01 05 54 65 78 74 0D A string...Text.
04BA:0110 0A
```

このようにASCII文字は，右側に示されます．

```
DB
```

この例では，現ダンプアドレスにつづく128バイトが表示されます．たとえば，前回のDumpコマンドの際の最終バイトが "04BA:0110" であった場合，このコマンドでは "04BA:0111" からバイトの表示が開始されます．

```
DB  table  table + 5
```

この例では，シンボルアドレス "table" から始まる6バイトが表示されます．

Dump Words

書式 DW [<アドレス> | <レンジ>]

機能 メモリ内容のワード（2バイト）単位の16進ダンプ

解説 Dump Wordsコマンドは，入力された〈アドレス〉，または〈レンジ〉のワード（2バイト値）の16進値を表示します．入力するアドレスまたはレンジによって，コマンドが1行または複数行を表示します．

各行には，その行の先頭ワードのアドレスが表示され，続いて最大8個の16進ワード値が表示されます．16進値は，スペースによって区切られます．コマンドは，レンジの終わりまで，または最初の64ワードが表示されるまで値の表示を行います．

例 DWコマンドの使用例をつぎに示します．

```
DW  CS:100  110
```

この例では，"CS:100" から "CS:110" までのワード値が表示されます．表示はつぎのようになります．

```
04BA:0100 2041 7473 6972 676E 0104 5405 7865 0A0D
04BA:0110 002E
```

1行につき8個より多くの値は表示されません．

```
DW
```

この例では，現ダンプアドレスの64ワードが表示されます．たとえば，前回のDumpコマンドの際の最終バイトが "04BA:0110" であった場合，このコマンドでは，"04BA:0111" からワードの表示が開始されます．

```
DW  table  table + 5
```

この例では，シンボルアドレス "table" から "table+5" までのレンジのワードが表示されます．

Dump Doublewords

書　式　DD [〈アドレス〉|〈レンジ〉]

機　能　**メモリ内容の倍長ワード(4バイト)単位のダンプ**

解　説　Dump Doublewordsコマンドは，入力された〈アドレス〉または〈レンジ〉の倍長ワード(4バイト値)の16進値を表示します．入力するアドレスまたはレンジによって，コマンドは1行から複数行を表示します．各行には，その行の最初の倍長ワードのアドレスが表示され，続いて最大4個の16進倍長ワード値が表示されます．各16進値は，スペースで区切られます．コマンドは，レンジが終わるまで，または最初の32個の倍長ワードを表示します．

例　DDコマンドの使用例をつぎに示します．

```
DD CS.100 110
```

この例では，〝CS:100〞から〝CS:110〞までの倍長ワードが表示されます．表示はつぎのようになります．

```
04BA:0100 7473:2041 676E:6972 5054:0104 0A0D:7865
04BA:0110 0000:002E
```

1行につき，4個より多くの値は表示されません．

```
DD
```

この例では，現ダンプアドレスの32個の倍長ワードが表示されます．たとえば，前回のDumpコマンドの最終バイトが04BA:0110であった場合，このコマンドでは，04BA:0111から倍長ワードの表示が開始されます．

```
DD table table + 5
```

この例では，シンボルアドレス〝table〞から〞table+5〞までのレンジの倍長ワードが表示されます．

Dump Short reals

書　式　DS [〈アドレス〉|〈レンジ〉]

機　能　短い（4バイト）の浮動小数点数の表示

解　説　Dump Short realsコマンドは，入力されたレンジの短い(4バイト)浮動小数点数の16進値と10進値を表示します．入力するアドレスまたはレンジによって，コマンドが1行または複数行を表示します．各行には，短い実数のアドレスが表示され，次にその数のバイトの16進値，その数の10進値が続きます．16進値は，スペースで区切られます．10進値の形式は，つぎのとおりです．

±.dd...dE ± mm

表示を行う場合，SYMDEBは表示を行う前に，短い実数を長い実数に変換します．そのため，最大16個の10進数を表示しますが，最初の7個の数字のみが有効です．

コマンドは，最低1個の値を表示します．レンジを入力すると，コマンドはそのレンジ内のすべての値を表示します．

例　DSコマンドの使用例をつぎに示します．

```
DS  ds:100
```

この例では，アドレス〝ds:100〟の短い実数が表示されます．表示はつぎのようになります．

```
04BA:0100 00 00 20 40 +.25E+1
```

このように，1行につき1個の値だけが表示されます．

```
DS  pi
```

この例では，シンボルアドレス〝pi〟の短い実数が表示されます．

Dump Long reals

書　式　DL　[〈アドレス〉|〈レンジ〉]

機　能　長い（8バイト）の実数の表示

解　説　Dump Long realsコマンドは，入力されたレンジの長い（8バイト）浮動小数点数の16進値と10進値を表示します．入力する〈アドレス〉または〈レンジ〉によって，コマンドが1行または複数行を表示します．

各行には，浮動小数点数のアドレスが表示され，次にその数のバイトの16進値，その数の10進値が続きます．16進値は，スペースで区切られます．10進値の形式は，つぎのとおりです．

±.dd...dE ± mm

最高16個の10進数が表示されます．

コマンドは，最低1個の値を表示します．〈レンジ〉を入力すると，その範囲内のすべての値を表示します．

例　DLコマンドの使用例をつぎに示します．

```
DL  ds:100
```

この例では，アドレス〝ds:100″の長い実数が表示されます．表示はつぎのようになります．

```
04BA:0100 86 37 6B F0 BE 2A 57 3F +.14139993273678774E − 2
```

このように，1行につき1個の値だけが表示されます．

```
DL  gamma
```

この例では，シンボルアドレス〝gamma″の長い浮動小数点数が表示されます．

Dump Ten-byte reals

書　式　DT [〈アドレス〉|〈レンジ〉]

機　能　10 バイトの長い浮動小数点数の表示

解　説　Dump Ten-byte reals コマンドは，入力された〈アドレス〉または〈レンジ〉の 10 バイトの浮動小数点数の 16 進値と 10 進値を表示します．

入力するアドレスまたはレンジによって，コマンドが 1 行か複数行を表示します．各行には，浮動小数点数のアドレスが表示され，次にその数のバイトの 16 進値，10 進値が続きます．16 進値は，スペースで区切られます．10 進値の形式はつぎのとおりです．

± . dd. . . dE ± mm

最大 16 個の 10 進数が表示されます．

コマンドは，最低 1 個の値を表示します．レンジを入力すると，コマンドはその範囲内のすべての値を表示します．

例　DT ds:100

この例では，アドレス〝ds:100〟の浮動小数点数が表示されます．表示はつぎのようになります．

```
04BA:0100 86 37 6B F0 BE 2A 57 3F 00 00 +.14139993273678774E-2
```

このように 1 行につき 1 個の数だけが表示されます．

DT gamma

この例では，シンボルアドレス〝gamma〟の 10 バイトの浮動小数点が表示されます．

Dump

書式　D ［<アドレス>｜<レンジ>］

機能　メモリ内容の16進とASCIIダンプ

解説　Dumpコマンドは，入力された〈アドレス〉または〈レンジ〉のメモリの内容を表示します．このコマンドは，直前に用いたDAコマンド，DSコマンド，DLコマンド，DTコマンドの定義する形式と同じ形式でメモリを表示します．

入力するアドレスまたはレンジによって，コマンドは1行か複数行を表示します．各行には，最初に表示される項目のアドレスが表示されます．コマンドは，最低1個の値を表示します．レンジを入力すると，コマンドはレンジ内のすべての値を表示します．アドレスやレンジを入力しないと，コマンドは，最後に表示されたバイトの次のバイトのメモリの内容を表示します．

注意

Dumpコマンドの名前は，最低1つのスペースで，〈アドレス〉値または〈リスト〉値と区切らなければなりません．

例　Dコマンドの使用例をつぎに示します．

```
DA ds:100
04BA:0100 A string..
D
04BA:010B Text...
```

この例では，Dumpコマンドは，Dump Asciiコマンドで表示された列の直後のアドレスのASCII文字を表示します．

```
DW ds:100 101
04BA:0100 2041
D ds:324 325
04BA:0324 FE31
```

この例では，Dumpコマンドは，アドレス〝ds:324″のワードを表示します。

```
DS pi
04BA:0100 00 00 20 40 +.25E+1
D gamma
04BA:0104 00 00 20 40 +.25E+1
```

この例では，Dumpコマンドは，シンボルアドレス〝pi″の短い実数を表示します。

Enter

書　式　E <アドレス> [<リスト>]

機　能　メモリへバイト値を入れる

解　説　Enterコマンドは，1個または複数のバイト値をメモリの指定されたアドレスに入れます．<リスト>を入力すると，コマンドは，リスト中のすべての値を使ってしまうまで，入力されたアドレスのバイトと，それに続くアドレスのバイトの置換を行います．<リスト>を省略したときは，コマンドは，バイト値を促します．

エラーが起きた場合，すべてのバイト値は，もとのままになっています．

<リスト>を入力しないと，SYMDEBは，アドレスとその現在値とピリオド(.)を表示して，そのアドレスの新しい値の入力を求めます．そこで，つぎに示す方法を用いて，バイト値を置き換えたり，次のアドレスに進んだり，また前のアドレスに戻ったり，コマンドを終了したりすることができます．

1. バイト値を置き換えるときは，現在の値の次に新しい値をタイプします．入力が16進数で1桁か2桁であることを確認してください．コマンドは，それ以上の余分な数字や文字を無視します．
2. 次のバイト（アドレス）へ進む場合には，スペースキーを押します．いったん次のバイトへ進むと，その値を変更したり，さらに次のバイトへ進むことができます．8バイトの境界を越えて飛び越すと，SYMDEBは新しいアドレスと値を表示して，新しい表示行を開始させます．
3. 前のバイト（アドレス）へ戻る場合は，ハイフン(-)をタイプします．前のバイトに戻すと，SYMDEBは，そのバイトのアドレスと値を用いて，新しい表示行を開始させます．
4. Eコマンドを終了するときは，リターンキーを押します．コマンドの終了はいつでも行うことができます．

例　Eコマンドの使用例をつぎに示します．

```
E CS:100 1 2B E5
```

この例では，CS:100と1，CS:101と2B，CS:102とE5の各々の間で，バイトの置換が行われます．

```
E ES:100
```

この例では，SYMDEB がつぎのようなプロンプトを表示します．

```
04BA:0100 EB.
```

ここで，41 とタイプすると，値 EB を新しい値 41(H)に変更できます．

```
04BA:0100 EB.41
```

スペースキーを押すと，次のバイトへ進みます．

```
04BA:0100 EB.41 10:
```

ハイフンをタイプすると，前の値へと戻ることができます．

```
04BA:0100 EB.41 10:-
04BA:0100 41.
```

Enter Bytes

書式　EB 〈アドレス〉 [〈リスト〉]

機能　指定したアドレスへバイト値を入れる.

解説　Enter Bytesコマンドは，1つまたは複数のバイト値をメモリに収めます．オプションの 〈リスト〉 に複数のバイト値が指定されているときは，その中の値が〈アドレス〉から始まるメモリに連続して収められます．

〈リスト〉をつけずに，アドレスだけが指定されたときは，アドレスとそのメモリの現在のバイト値とピリオド(.)が表示されます(前項，Eコマンド同様)．ユーザーはこの時点で，メモリに新しいバイト値を入れる，前後のアドレス(バイト)に移るなどの操作を行うことができます．

- 表示されているバイト値を変更するときは，その値をタイプします．ピリオドの右側に入力したバイト値が表示されます．
- 次のバイト値（アドレス）に進むときは，スペースキーを押します．SYMDEBは，スクリーンの1行に8バイトずつ表示を行い，アドレスが8バイト分以上進むと，新しい行にその行の先頭アドレスとバイト値を表示します．
- 前のアドレスに戻るときは，ハイフン(-)をタイプします．新しい行に，そのバイトのアドレスと，現在値が表示されます．
- EBコマンドを終了して，SYMDEBのプロンプトに戻るときは，リターンキーを押します．
- 〈リスト〉で，ストリングス（引用符で囲まれた文字）が指定されると，前項のEAコマンドと同様に，文字（文字列）のASCIIコード値が収められます．

例　EBコマンドの使用例をつぎに示します．

```
-EB  CS:100  01  2B  E5
-
```

この例は，CS :100番地から始まる3バイト（CS:100，CS:101，CS:102)に，新しいバイト値，〝01″，〝2B″，〝E5″を入れたものです．

```
-EB  CS:100
```

```
2344:0100    F3.
```

この例は，アドレスだけを指定した場合のものです．SYMDEB は，アドレス〝2344：0100″とそのメモリの現在の値〝F 3″を表示します．

この値を〝5 E″に変更するときは，新しい値をタイプします．

```
-EB  CS:100
2344:0100  F3.5e
```

次のバイト値に進むときは，スペースキーを押します．

```
-EB  CS:100
2344:0100  F3.5e      10.
```

ここにも新しいバイト値〝76″をタイプします．

```
-EB  CS:100
2344:0100  F3.5e      10.76      B0.
```

先に入力した値〝76″は間違いで，正しくは〝77″であったとします．前のアドレスに戻るときは，ハイフンをタイプします．

```
-EB  CS:100
2344:0100  F3.5e      10.76      B0.-
2344:0101  76.
```

正しい値を入力しなおします．

```
-EB  CS:100
2344:0100  F3.5e      10.76    B0.-
2344:0101  76.77
```

必要な入力が済んで，EB コマンドを終了するときは，リターンキーを押します．

```
-EB  CS:100
2344:0100  F3.5e      10.76      B0.-
2344:0101  76.77
```

SYMDEB のプロンプトが表示され，つぎの SYMDEB コマンドを入力できるようになります．

Enter Ascii

書　式　EA 〈アドレス〉 [〈リスト〉]

機　能　**指定したアドレスへリスト中の文字のASCIIコード値を入れる.**

解　説　Enter Ascii コマンドは，事項で解説するEBコマンドと同一の働きをします．〈リスト〉中の文字（文字列）のASCIIコード値が指定したアドレスから始まるメモリに収められます．

例　EAコマンドの使用例をつぎに示します．

```
-EA  data_seg:msg2  "Can't open file"
-
```

この例では，文字列"Can't open file"のASCIIコードの値が，アドレス"data_seg:msg2" 以降に収められます．

Enter Words

書 式　EW 〈アドレス〉 [〈値〉]

機 能　**指定したアドレスへ，ワード値を入れる**

解 説　Enter Wordsコマンドは，指定したアドレスのメモリへ，ワード値(2バイト)を収めます．〈値〉で指定できる値は，1回に1ワード分です．

〈値〉をつけずに，アドレスだけが指定されると，SYMDEBは，そのアドレスから始まる1ワード(2バイト)分のメモリの現在の値を表示し，ユーザーの入力を待ちます．

〈値〉を指定したときは，そのアドレスの1ワードを変更し，つづいて次のアドレスとそのワード値を表示して，ユーザーの入力を待ちます．

EWコマンドを終了して，SYMDEBのコマンド入力モードに戻るときは，リターンキーを押します．

例　EWコマンドの使用例をつぎに示します．

```
-EW  CS:400    4e3a
2344:0402      ED32.8ad8
2344:0404      1D3C.
-
```

この例は，アドレス"CS:400"の1ワード（0400と0401）にワード値"4E3A"を入れるものです．EWコマンドは，このワードに新しいワード値を収め，つづいて次のアドレス"2344:0402"とその内容"ED 32"を表示します．この例では，この新しいワードに"8 ad 8"を入れています．EWコマンドは，さらに次のアドレスとそのワード値を表示しています．このような状態でリターンキーを押すと，EWコマンドは終了し，SYMDEBのコマンド入力モードに戻ります．

Enter Doublewords

書 式　ED 〈アドレス〉 [〈値〉]

機 能　指定したアドレスへ，2ワード値を入れる

解 説　Enter Doublewordsコマンドは，指定したアドレスへ，2ワード(4バイト)の値を入れます．〈値〉で指定できる値は，1回に2ワード分です．

〈値〉をつけずに，アドレスだけを指定すると，EDコマンドは，そのアドレスと現在のメモリ値を表示して，ユーザーの入力を待ちます．

〈値〉を指定すると，指定したアドレスに2ワード値を収めて，続きのアドレスとその現在値を表示し，ユーザーの入力を待ちます．

EDコマンドを終了して，SYMDEBのコマンド入力モードに戻るときは，リターンキーを押します．

例　EDコマンドの使用例をつぎに示します．

```
-ED  CS:100       12 EF  :CD 01
2344:0104         440 E  :1234 .1234:5678
2344:0108         8 ED 9:1234
 -
```

この例は，"CS:100" から始まる2ワードに "12 EF:CD 01" を入れるものです．EDコマンドは，この値を入れた後，続きのアドレス "2344:0104" とその現在値 "440 E:1234" を表示します．ユーザーはつづいて2ワード値 "1234:5678" を入力しています．EDコマンドは，さらに続きのアドレスと2ワード値を表示します．このような状態で，リターンキーを押すと，SYMDEBのコマンド入力モードに戻ります．

Enter Short reals

書 式　ES 〈アドレス〉 [〈値〉]

機 能　**指定したアドレスへ，短い実数（4バイト）を入れる**

機 能　Enter Short reals コマンドは，指定したアドレスへ4バイト形式の浮動小数点値（実数）を入れます．〈値〉で指定できる短い実数は，1回に1個です．

〈値〉をつけずにアドレスだけを指定すると，ES コマンドは，そのアドレスから始まる4バイト分のメモリの現在値を浮動小数点表示で示し，ユーザーの入力を待ちます．

〈値〉を指定すると，ES コマンドは，指定されたアドレス（4バイト分）の内容を変更し，続いて次のアドレスとその現在値を表示し，ユーザーの入力を待ちます．

ES コマンドを終了して，SYMDEB のコマンド入力モードに戻るときは，リターンキーを押します．

例　ES コマンドの使用例をつぎに示します．

```
-ES  pi  3.1415926
-
```

この例は，シンボリックアドレス"pi"へ，3.1415926 という数値を短い実数の形式で入れたものです．ES コマンドで，アドレスだけを指定した場合は，つぎのようになります．

```
-ES  pi
210C:0130    -0.1256210825216E+16    +0.31415926e+1
210C:0134    -0.4309309980615894E-31
```

ES コマンドで入れた値を調べるには，DS コマンドを用います．先頭バイトのアドレス，4バイトの内容，浮動小数点形式で表示した10進数の数値が表示されます．この数値は，16桁で表示されますが，有効数字は小数点以下7桁までで，後ろの9桁は意味を持ちません．

```
-DS  pi
210C:0130  DA  0F  49  40  +0.3141592502593994E+1
-
```

Enter Long reals

書式 EL 〈アドレス〉 [〈値〉]

機能 指定したアドレスへ，長い実数（4バイト）を入れる．

機能 Enter Long reals コマンドは，指定したアドレスへ8バイト形式の浮動小数点値（実数）を入れます．〈値〉で指定できる長い実数は，1回に1個です．

〈値〉をつけずにアドレスだけを指定すると，ELコマンドは，そのアドレスから始まる8バイト分のメモリの現在値を浮動小数点表示で示し，ユーザーの入力を待ちます．

〈値〉を指定すると，ELコマンドは，指定されたアドレス（8バイト分）の内容を変更し，続いて次のアドレスとその現在値を表示し，ユーザーの入力を待ちます．

ELコマンドを終了して，SYMDEBのコマンド入力モードに戻るときは，リターンキーを押します．

例 ELコマンドの使用例をつぎに示します．

```
-EL  pi  3.141592653589793
-
```

この例は，シンボリックアドレス"pi"に，数値"3.141592653589793"を長い実数の形式で入れたものです．ELコマンドで，アドレスだけを指定した場合は，つぎのようになります．

```
-EL  170
210C:0170  +0.1343280735843091 E+65299  +0.3141592653589793 e+1
210C:0178  +0.1040230032441619 E-71
-
```

Enter Ten-byte reals

書 式　ET 〈アドレス〉 [〈値〉]

機 能　指定したアドレスへ，10バイト実数を入れる

機 能　Enter Ten-byte reals コマンドは，指定したアドレスへ10バイト形式の浮動小数点値（実数）を入れます．〈値〉で指定できる短い実数は，1回に1個です．

〈値〉をつけずにアドレスだけを指定すると，ETコマンドは，そのアドレスから始まる10バイト分のメモリの現在値を浮動小数点表示で示し，ユーザーの入力を待ちます．

〈値〉を指定すると，ETコマンドは，指定されたアドレス（10バイト分）の内容を変更し，続いて次のアドレスとその現在値を表示し，ユーザーの入力を待ちます．

ETコマンドを終了して，SYMDEBのコマンド入力モードに戻るときは，リターンキーを押します．

例　ETコマンドの使用例をつぎに示します．

```
-ET pi 3.141592653589793
-
```

この例は，シンボリックアドレス"pi"へ，数値"3.141592653589793"を入れたものです．ETコマンドで，アドレスだけを指定した場合は，つぎのようになります．

```
-ET pi
210C:0150  +0.0204654128113587E+7898  +0.314152653589793e+1
210C:015A  +0.5976239733286124E+3896
```

eXamine symbol map

書　式　X　[＊]

X?　[〈マップ名〉!] [〈セグメント名〉:] [〈シンボル名〉]

機　能　**シンボル名とアドレスのリスト表示**

解　説　eXamine symbol mapコマンドは，現シンボルマップ内のシンボルの名前とアドレスを表示します．SYMDEBは，SYMDEBコマンド行に入力された各シンボルファイル名に対して，シンボルマップを作成します．そこで，eXamine symbol mapを用いて，マップの内容を調べることができます．

Xコマンドは，現シンボルマップとそのマップ内のセグメントの名前及びロードセグメントアドレスを表示します．アスタリスク（＊）を指定すると，コマンドはすべてのシンボルマップの名前とロードセグメントアドレスを表示します．

X?コマンドは，シンボルマップ内の1つまたは複数のシンボルの名前とアドレスを表示します．

“〈マップ名〉!”を入力すると，コマンドはそのシンボルマップの情報を表示します．マップ名は，対応するシンボルファイルのファイル名（拡張子なし）でなくてはなりません．

“〈セグメント名〉:”を入力すると，コマンドはそのセグメントの名前とロードセグメントアドレスを表示します．セグメント名は，シンボルマップ内か，現在開かれているシンボルマップ内に挙げられている正しいセグメントの名前でなければなりません．

“〈シンボル名〉”を入力すると，コマンドは，そのシンボルのセグメントアドレスとセグメントのオフセットを表示します．シンボル名は，入力されたセグメント内のシンボルの名前でなければなりません．

複数のセグメントやシンボルの情報を表示する場合，セグメント名やシンボル名にアスタリスク(＊)をつけることができます．アスタリスクは，ワイルドカード的な文字として働くので，SYMDEBは，セグメント名やシンボル名の始まりと同じ文字で始まる名前のすべてのセグメントやシンボルに関する情報を表示します．たとえば，“F＊:”は，文字“F”で始まるすべてのセグメント名であり，“＊”は，アンダースコア（_）で始まるすべてのシンボルです．

例 Xコマンドの使用例をつぎに示します.

```
X
```

この例では, 現シンボルマップの名前と, そのマップ内のセグメントの名前と, ロードセグメントアドレスが表示されます.

```
X?  test!
```

この例では, シンボルマップファイル"test"のロードセグメントアドレスが表示されます.

```
X?  igroup:
```

この例では, 現在オープンしているシンボルマップ内の"igroup"という名前のセグメントのロードセグメントアドレスが表示されます.

```
X?  test!igroup:
```

この例では, シンボルマップ "test" 内の "igroup" という名前のセグメントのロードセグメントアドレスが表示されます.

```
X?  start
```

この例では, 現在オープンしているシンボルマップ内のシンボル "start" のセグメントのアドレスと, セグメントのオフセットが表示されます.

```
X?  test!start
```

この例では, シンボルマップ "test" 内のシンボル "start" のセグメントのアドレスとセグメントのオフセットが表示されます.

```
X?  test!igroup: start
```

この例では, シンボルマップ"test"内のセグメント"igroup"のシンボル"start"のセグメントのアドレスと, セグメントのオフセットが表示されます.

```
X?  code *
```

この例では, 現シンボルマップ内の文字 "code" で始まるすべてのシンボルのセグメントのアドレス値とオフセット値が表示されます.

Fill

書　式　F 〈レンジ〉〈リスト〉

機　能　**〈レンジ〉のメモリを〈リスト〉の値で満たす**

解　説　Fillコマンドは，入力された〈レンジ〉内のアドレスのメモリを，〈リスト〉の値で満たします．〈リスト〉の長さが〈レンジ〉よりも短いときは，レンジ内のすべてのバイトが満たされるまで，〈リスト〉が繰り返して用いられます．〈リスト〉の値が〈レンジ〉のバイト数より大きい場合，コマンドは余分な値を無視します．

例　Fコマンドの使用例をつぎに示します．

```
F  CS:100  L  100  FF
```

この例では，メモリのCS:100からCS:1FFまでがバイト値FFHで満たされます．

```
F  DGROUP:table  L  64  42  45  52  54  41
```

この例では，“DGROUP:table”から100（64H）バイトが，入力されたバイト値で満たされます．前記の5個の値は，100バイトすべてが一杯になるまで繰り返されます．

Go

書 式　G ［＝開始アドレス］［ブレークポイントアドレス］

機 能　デバッグ中のプログラムの実行

解 説　Goコマンドは，入力された開始アドレスのプログラムに実行の制御を渡します．実行はプログラムが終わるまで，またはブレークポイントアドレスにくるまで続きます．また，プログラムは，Breakpoint Setコマンドを用いてセットされたブレークポイントで停止します．

開始アドレスを省略すると，コマンドは，CSレジスタとIPレジスタの現在値が指定するアドレスへから実行始めます．開始アドレスを指定するときは，イコール記号（＝）をつけます．

ブレークポイントアドレスは，命令オペコードの最初のバイトを指定します．1度に最高10個のブレークポイントアドレスを，任意の順序で入力できます．実行の最初にくるアドレスのみが，ブレークポイントを発生させ，他はすべて無視されます．10個より多いブレークポイントをセットしようとすると，SYMDEBはエラーメッセージを表示します．

プログラムの実行の際にブレークポイントに達すると，SYMDEBはすべてのレジスタとフラグの現在値を表示します．また，次に実行する命令の表示も行います．表示の形式は，Registerコマンドと同じです．

注意

Goコマンドは，IRET命令を用いて，プログラムに制御を渡します．そのため，Goコマンドはユーザーのスタックポイントをセットして，ユーザーフラグとCSレジスタ及びIPレジスタをユーザースタックに入れなくてはなりません．ユーザーのスタックが6バイトなかったり，無効なメモリ内にある場合，Goコマンドがオペレーティングシステムの故障を引き起こすことがあります．

ブレークポイント作成の際，SYMDEB　は，各ブレークポイントアドレスにINT 3命令（割り込みコード0CCH）を置き，ブレークポイントがくると，このアドレスを元の命令に復元します．しかし，実行がプログラムの最後まで続けられたり，何か他の理由で中止された場合，SYMDEBは，割り込みコードの置換は行いません．このため，プログラムを再開させる前にNameコマンドとLoadコマンドを用いて，プログラムを再ロードしなくてはなりません．

プログラムが最後まで実行された場合，SYMDEB は〝Program terminated normally″というメッセージを表示します．

例　　Gコマンドの使用例をつぎに示します．

```
G=CS:0    CS:7550
```

この例では，アドレス〝CS:0″のプログラムに実行の制御が渡されます．SYMDEBは，ブレークポイントアドレス〝CS:7550″の命令がくると，実行を中止して，レジスタとフラグの現在値を表示します．

```
G
```

この例では，CSレジスタとIPレジスタの現在値が示す命令へ制御が渡されます．このコマンドは一般に，ブレークポイント後の実行の継続に用いられます．

```
G=_main    add
```

この例では，シンボルアドレス〝_main″が指名する命令に制御が渡されます．ブレークポイントは，アドレス〝add″にセットされます．

Help

書式 ?

機能 SYMDEB コマンドの一覧表示

解説 Help コマンドは，SYMDEB コマンドのリストを表示します．表示されるリストをつぎに示します．

```
-?
A [<address>] - assemble                M <range> <address> - move
BC[<bp>] - clear breakpoint(s)          N <filename> [<filename>...] - name
BD[<bp>] - disable breakpoint(s)        O <value> <byte> - output to port
BE[<bp>] - enable breakpoint(s)         P [=<address>] [<value>] - program step
BL[<bp>] - list breakpoint(s)           Q - quit
BP [bp] <address> - set breakpoint      R [<reg>] [[=] <value>] - register
C <range> <address> - compare           S <range> <list> - search
D[type][<range>] - dump memory          S (-|&|+) - source level debugging
E[type] <address> [<list>] - enter      T [=<address>] [<value>] - trace
F <range> <list> - fill                 U [<range>] - unassemble
G [=<address> [<address>...]] - go      V [<range>] - view source lines
H <value> <value> - hexadd              W [<address> [<drive><rec><rec>]] - write
I <value> - input from port             X [?] <symbol> - examine symbols(s)
K [<value>] - stack trace               XO<symbol> - open map/segment
L [<addr> [<drive><rec><rec>]] - load Z <symbol> <value>

? <expr> - display expression           > ) <device/file> - Redirect output
! [dos command] - shell escape          < ( <device/file> - Redirect input
. - display current source line         = ~ <device/file> - Redirect both
                                        * <string> - comment

<expr> ops: + - * / : not seg off by wo dw poi port wport mod and xor or
<type> : Byte, Word, Doubleword, Asciz, Shortreal, Longreal, Tenbytereal
```

Hex

書　式　H 〈数値1〉〈数値2〉

機　能　16進数の和と差の計算

解　説　Hexコマンドは，2つの16進数の和と差を表示します．SYMDEBは，〝数値1〞と〝数値2〞を加算して結果を表示し，次に〝数値1〞から〝数値2〞を引いてその結果を表示します．結果は，常に1行に16進数で示されます．一般的な式を評価する場合は，〝Displayコマンド〞の項を参照してください．

例　Hコマンドの使用例をつぎに示します．

```
H 3 4
```

この例では，7とFFFFという結果が表示されます．

```
H 110 100
```

この例では，210と10という結果が表示されます．

Input

書 式　I 〈ポート〉

機 能　**指定した〈ポート〉から1バイト読み込む**

解 説　Input コマンドは，入力された入力ポートから1バイトを読み込んで表示します．

例　I コマンドの使用例をつぎに示します．

```
I 2F8
```

この例では，入力ポート番号“2F8”から読み取ったバイト値が表示されます．

Load

書　式　L [〈アドレス〉 | [〈ドライブ〉 〈レコード〉 〈カウント〉]]

機　能　**ファイルまたはディスクの論理レコードを読み込む**

解　説

Loadコマンドは，Nameコマンドで指定されたファイルや，入力された数の論理ディスクレコードの内容をメモリに読み込みます．内容は，入力されたアドレスかデフォルトのアドレスにコピーされ，BX:CXレジスタのペアにロードバイト数がセットされます．

ファイルをロードするときは，Loadコマンドを使用する前に，ファイル名を入力しなくてはなりません．ファイル名は，Nameコマンドを用いて入力できますが，SYMDEBの起動時に，名前をプログラムのアーギュメントとして通しても入力できます．

ファイル名を入力しないと，Loadコマンドは，DSがDSレジスタの現在値である場所，DS:5Cの現在名を使用します．この場所は，Nameコマンドを用いて，入力されたファイル名やプログラムのアーギュメントとして通されたファイル名を受け入れます．

〈アドレス〉を指定すると，ファイルやセクタの内容はこのアドレスから始まるメモリに入れられ，〈アドレス〉を指定しなければCS:100から始まるメモリに入れられます．CSはCSレジスタの現在値です．

ディスクから直接論理レコードをロードするときは，〈アドレス〉，〈ドライブ〉，〈レコード〉，〈カウント〉の値を明確に入力します．ドライブは，読み取るべきドライブの名前で，ドライブA：(0)，ドライブB：(1)，ドライブC：(2)，ドライブD：(3) を表します．レコードは，ドライブから読み取る最初の論理レコード番号で，1〜4桁の16進数です．〈カウント〉は，ディスクから読み取るレコード数を指定します．カウントは，1〜4桁の16進数です．

注意

指定したファイルが.EXE形ファイル（拡張子〝.EXE″をもつ）である場合，Loadコマンドはロードアドレスを.EXEファイルのヘッダに入力されたアドレスに直します．つまり，.EXEファイルの場合には，〈アドレス〉の指定は無効です．

Loadコマンドは，ロードを行う前に，.EXEファイルのヘッダを取り除きます．したがって，実際にロードされたバイト数は，.EXEファイルのバイト数と食

い違います．

指定したファイルが .COM 形ファイル（拡張子〝.COM″をもつ）である場合，必ず CS:100 からロードされますので，〈アドレス〉を指定しても無効となります．

指名されたファイルに〝.HEX″拡張子がある場合，Load コマンドは，ファイルのロードを行う前に，そのファイルの開始アドレスとアドレスを加算します．アドレスを入力しないと，ファイルはその開始アドレスにロードされます．

例　L コマンドの使用例をつぎに示します．

```
N  file.com
L
```

この例では，〝file.com″という名前のファイルが，メモリのアドレス CS:100 にロードされます．ロードされたバイト数は，BX:CX レジスタペアに収められます．

```
L  DGROUP:table
```

この例では，シンボルアドレス〝DGROUP:table″から始まる記憶位置にファイルがロードされます．コマンドは，DS:5C の位置の現在のファイル名を使用します．

```
L  workspace  2  34  3
```

この例では，論理レコード番号 52（34H）から始まる 3 つの論理レコードがメモリのシンボルアドレス〝workspace″へロードされます．

Move

書式　M　〈レンジ〉〈アドレス〉

機能　**〈レンジ〉で指定したメモリブロックを〈アドレス〉へ移動（コピー）**

解説　Moveコマンドは，〈レンジ〉の指定するメモリのブロックを，〈アドレス〉から始まる位置へ移動させます．

すべての移動は，ソースブロックと宛先ブロックが重なってもデータを損失することなく遂行されます．〈レンジ〉で指定したアドレスのメモリは，Moveコマンド実行後もそのまま残っています．ただし，宛先ブロックがソースブロックの一部と重なった場合は，元のソースが変更されます．

データの損失を防ぐ為に，Moveは，ソースが宛先よりも高位のアドレスにある場合，最初にソースブロックの最も低位のアドレスからデータをコピーします．ソースが低位のアドレスにある場合，Moveは，最初にソースの最も高位のアドレスからデータをコピーします．

例　Mコマンドの使用例をつぎに示します．

```
M  CS:100  110  CS:500
```

この例では，CS:100からCS:110までのレンジのすべてのバイトが，CS:500から始まるメモリに移動します．

```
M DS:table L 100 workspace
```

この例では，シンボルアドレス〝DS:table〟の256（100H）バイトが，アドレス〝workspace〟へコピーされます．

Name

書 式　N [<ファイル名>] [<引数>]

機 能　ファイル名，引数のセット

解 説　Name コマンドは，以後の Load コマンドや Write コマンドのためにファイル名をセットしたり，以後のロード済プログラムの実行のためにプログラムの引数をセットしたりします．

ファイル名を入力すると，以後のすべての Load コマンドや Write コマンドは，ディスクファイルのアクセスの際にこの名前を使用します．

引数を入力すると，コマンドは，スペースを含むすべての引数を DS:81 から始まる記憶位置にコピーして，DS:80 のバイトをコピーされた全文字数にセットします．どちらの場合も，DS は DS レジスタの現在値です．引数をいったんコピーすれば，デバッグ中のプログラムによるアクセスが可能となります．

注意

最初の 2 つの引数がファイル名である場合，コマンドは，アドレス DS:5C とアドレス DS:6C にファイルコントロールブロックを作成して，このブロックに名前（正しい形式の名前）をコピーします．これによって，デバッグ中のプログラムがファイルコントロールブロック（FCB）を使用することができます．

Name コマンドは，ファイル名を引数としても扱い，DS:81 へコピーしたり，DS:5C にファイル名のファイルコントロールブロックを作成したりします．したがって，Load や Write 用に新しいファイル名をセットすると，前のプログラム引数は破壊されてしまいます．

Name コマンドは，つぎに示すアドレスのメモリを変更したり破壊したりすることがあります．

DS:5C	ファイル 1 のファイルコントロールブロック
DS:6C	ファイル 2 のファイルコントロールブロック
DS:80	文字の総数
DS:81	タイプされたすべての文字

例　　Nコマンドの使用例をつぎに示します．

```
N  file.exe
```

この例では，以後のLoadコマンドやWriteコマンドの為に，ファイル名が〝file 1.exe〟にセットされます．

```
N  file2.dat  file3.dat  /m  /b
```

この例では，デバッグ中のプログラムのためにプログラムの引数がセットされます．コマンドは，ファイル〝file2.dat〟とファイル〝file3.dat〟用のファイルコントロールブロックを作成します．また，コマンドの頭文字〝N〟以外のコマンドライン全体をDS:81にコピーします．

Open Map

書 式　XO ［<マップ名>！］<セグメント名>

機 能　シンボルマップまたはセグメントをセット

解 説　Open Map コマンドは，アクティブシンボルマップとセグメント，またはそのどちらかをセットします．

〝<マップ名>!〞を入力すると，コマンドは，アクティブシンボルマップを入力されたマップにセットします．<マップ名>は，SYMDEB コマンド行に指定したシンボルファイルの中の 1 つのファイル名（拡張子なし）でなくてはなりません．

<セグメント>名を入力すると，コマンドは，指名されたセグメントにアクティブセグメントをセットします．<セグメント名>は，指定されたシンボルマップ内のセグメントの名前でなければなりません．

例　XO コマンドの使用例をつぎに示します．

```
XO   test!
```

この例では，シンボルマップ〝.test〞がアクティブになります．

```
XO   test!igroup
```

この例では，シンボルマップ〝test〞のセグメント〝igroup〞がアクティブになります．

```
XO   dgroup
```

この例では，現シンボルマップのセグメント〝dgroup〞がアクティブになります．

Output

書　式　O 〈ポート〉 〈バイト〉

機　能　ポートへの出力

解　説　Outputコマンドは，入力されたバイトを指定された出力ポートに送ります．〈ポート〉は，16ビットのポートアドレスです．

例　Oコマンドの使用例をつぎに示します．

```
O 2F8 4F
```

この例では，バイト値4F（16進）が出力ポート2F8へ送られます．

```
O 3 21
```

この例では，バイト値21（16進）が出力ポート3に送られます．

PTrace

書　式　P [=〈開始アドレス〉] [〈カウント〉]

機　能　割り込みにも対応したトレース

解　説　PTraceコマンドは，入力された開始アドレスの命令を実行してから，すべてのレジスタとフラグの現在値を表示します．表示の形式は，Registerコマンドと同じです．

〈開始アドレス〉を指定すると，PTraceコマンドはそのアドレスから実行を開始します．そうでなければ，現在のCSレジスタとIPレジスタが示す命令から実行を開始します．イコール記号（=）は，開始アドレスを入力する場合にのみ用いることができます．

〈カウント〉を指定すると，コマンドは，その数の命令を実行するまで停止しません．コマンドは，次の命令を実行する前に，個々の命令をレジスタやフラグの現在値を表示します．

注意

PTraceコマンドは，あらゆる呼び出しやソフトウェアの割り込みからの復帰や実行を自動的に行うこと以外は，Traceコマンドと同じです．Traceコマンドは，呼び出しや割り込みの実行後，常に停止して，実行の制御を呼び出されたルーチン内に残します．

例　Pコマンドの使用例をつぎに示します．

```
P = _main
```

この例では，"_main"の命令が実行され，レジスタとフラグの現在値が表示されます．また，次に実行する命令も表示されます．

```
P
```

この例では，現CSレジスタとIPレジスタが示す命令が実行されます．

Quit

書　式　Q

機　能　SYMDEB の終了

解　説　Quit コマンドは，SYMDEB ユーティリティを終了させ，MS-DOS に制御を戻します．

例　Q コマンドは，引数なしで使用します．

```
Q
```

この例では，SYMDEB が終了されます．

Redirection

書　式　＜装置名
＞装置名
＝装置名

機　能　入出力の切り換え

解　説　Redirectionコマンドは，コマンドの入力と出力を装置名で指名された装置へ切り換えます．

＜コマンドによって，SYMDEBは，指定された装置からすべてのコマンド入力を読み込みます．

＞コマンドによって，SYMDEBは，指定された装置へすべてのコマンド出力を書き出します．

＝コマンドによって，SYMDEBは，指定された装置と読み込みと書き出しを行います．

装置名には，たとえば，つぎのようなものがあります．詳しくは「ユーザーズリファレンスマニュアル」を参照してください．

AUX	補助入出力装置
CON	コンソール

補助入出力装置（AUX）を用いる場合，ポートのボーレート他のモードを，接続するターミナルに正確にセットしなくてはなりません．

一般に，Redirectionコマンドは，スクリーン画面を全面的に使用するプログラムをデバッグする場合に用います．

注意

入力をコンソール以外に切り換えると，CTRL＋Sキーと CTRL＋Cキーは無効となります．

例　Redirectionコマンドの使用例をつぎに示します．

```
> AUX
```

この例ではSYMDEBコマンドの出力が“AUX”の装置へと切り換えられます．

```
= AUX
```

この例では，コマンドの入力と出力が“AUX”へ切り換えられます．

Register

書　式　R　[<レジスタ名> [<数値>]]

機　能　レジスタ値の表示

解　説　Registerコマンドは，CPUのレジスタの内容を表示し，その内容の新しい値への変更を可能にします．

<レジスタ名>を入力しないと，コマンドは現在のCSレジスタとIPレジスタの値が示すアドレスのすべてのレジスタ，フラグ，命令を表示します．

<レジスタ名>を入力すると，コマンドは入力されたレジスタの現在値を表示し，新しい値を催促します．レジスタ名と値の両方を入力すると，コマンドはレジスタを入力された値に変更します．

<レジスタ名>は，つぎに示すうちの1つです．

AX	BP	SS
BX	SI	CS
CX	DI	IP
DX	DS	PC
SP	ES	F

IPとPCは同じレジスタ，つまり命令ポインタを指名するものです．Fは，フラグレジスタの名前です．

レジスタの値を変更する場合は，Registerコマンド入力時にレジスタの名前と値を入力します．同様に値を入力しないと，コマンドは，レジスタ名，その現在値，そしてコロン(:)のプロンプトを表示します．新しい値をタイプしてリターンキーを押してください．値を変更したくない場合は，リターンキーを押すだけで構いません．不適切なレジスタ名を入力すると，コマンドは"Bad Register"というメッセージを表示します．

フラグの値を変更する場合は，Registerコマンド入力時にレジスタ名"F"を入力します．コマンドは，各フラグの現在値を2文字略号で表示します．　フラグ値の略号のリストをつぎに示します

フラグ	セット	クリア
オーバーフロー	OV	NV
方向	DN（減少）	UP（増加）
割り込み	EI（有効）	DI（無効）
符号	NG（負）	PL（正）
ゼロ	ZR	NZ
補助桁上げ	AC	NA
パリティ	PE（偶数）	PO（奇数）
桁上げ（キャリー）	CY	NC

コマンドは，値のリストの終わりにハイフン (-) を表示します．ハイフンが示されたら，変更したいフラグの新しい値を入力して，リターンキーを押します．フラグの値の入力順序は任意です．値の間にスペースは必要ありません．新しい値の入力されないフラグは，変更されません．どのフラグも変更したくない場合は，リターンキーを押します．

1つのフラグに対して複数の値を入力すると，コマンドは〝Double flag!〟のメッセージを表示します．前記以外の名前を入力すると，コマンドは，〝Bad Flag!〟のメッセージを表示します．どちらの場合も，エラーが発生するまでのフラグの変更は行われますが，エラーのフラグとそれ以後のフラグは変更されません．

例

Rコマンドの使用例をつぎに示します．

```
R
```

この例では，CSレジスタとIPレジスタの指すアドレスの命令に加えて，すべてのレジスタとフラグの値が表示されます．表示はつぎのようになります．

```
AX = 0E00 BX = 00FF CX = 0007 DX = 01FF SP = 039D BP = 0000
SI = 005C DI = 0000 DS = 04BA ES = 04BA SS = 04BA CS = 04BA
IP = 011A NV UP DI NG NZ AC PE NC
04BA:011A CD21  INT     21
```

命令は最後に表示されます．

```
R   AX
```

この例では，AXレジスタの現在値と，新しい値に対するプロンプトが表示されます．表示はつぎのようになります．

AX　0E00

:

コロン（:）の後に16ビットの値をタイプします．変更したくない場合には，リターンキーを押します．

RF

この例では，現在のフラグ値と変更を求めるプロンプトが表示されます．表示はつぎのようになります．

NV　UP　DI　NG　NZ　AC　PE　NC　-

フラグ値の変更には，プロンプトの方法を用いなければなりません．コマンド行の値は，すべて無視されます．

R　IP　100

この例では，IPレジスタが値100に変更されます．

Search

書　式　　S 〈レンジ〉 〈リスト〉

機　能　　バイト値の検索（サーチ）

解　説

Searchコマンドは，入力されたレンジのメモリを捜して，リストで入力されたバイト値を見つけます．リストのバイトを見つけると，コマンドは，リストの発生アドレスを表示します．そうでない場合は，何も表示せず，SYMDEBのプロンプトが再び表示されます．

リストのバイト数はいくつでもよく，各バイトはスペースかカンマで区切らなければなりません．リストが複数のバイトを持っている場合，あるアドレスから始まるバイトが，リストのバイトの値及び順序に一致しない限り，Searchはそのアドレスを表示しません．

例

Sコマンドの使用例をつぎに示します．

```
S  CS:100  200  41
```

この例では，バイト値41を含むCS:100からCS:200の範囲の各メモリのアドレスが表示されます．

```
S  table  L  100  "Fix  up"
```

この例では，文字列"Fix up"を含む各メモリのアドレスが表示されます．Sコマンドは，シンボルアドレス "table" から始まる256（100H）バイトの範囲を捜します．

Set Source Mode

書 式　S {− | & | +}

機 能　命令コードの表示形式の設定

解 説　Set Source Modeコマンドは，命令（インストラクション）コードを表示するコマンドの表示形式をセットします．プラス記号(＋)を入力すると，SYMDEBは，表示すべき命令（インストラクション）に対応する実際のプログラムのソース行を表示します．マイナス記号(−)を入力すると，SYMDEBはメモリ内の命令を分解して表示します．アンパーサンド(&)を入力すると，SYMDEBは，実際のプログラムのソース行と，命令コードの両方を表示します．

Set Source Modeコマンドは，Registerコマンド，PTraceコマンド，TraceコマンドおよびUnassembleコマンドが表示する命令に影響を与えます．まず，SYMDEBは，命令コードのみを表示します．S＋コマンドかS&コマンドが入力されていて，さらに行番号の情報を持つシンボルファイルがロードされている場合に限って，SYMDEBはソース行を表示します．シンボルファイルをロードしていなかったり，シンボルファイルに行番号の情報が入っていないと，SYMDEBは，以後のソース行表示の要求を無視します．S&コマンドを入力すると，SYMDEBは，CS:IPが指定する現在の命令のアドレスが行番号と一致する場合だけ，ソース行を表示します．

ソース行の形式はつぎのとおりです．

```
行番号:ソース
```

ソース行は常に，命令コードが表示される前に表示されます．要求された行の表示の為に，SYMDEBが現ソースファイルを変更する必要がある場合，SYMDEBはその行を表示する前に新しいソースファイルの名前を表示します．

注意

SYMDEBは，シンボルファイルの行番号の情報を用いてソース行を表示しますが，これらの行を表示するには，ソースファイルの実際のファイル名に対するプロンプトが必要になります．SYMDEBが初めてソースファイルをアクセスする場合，つぎのプロンプトを表示します．

```
External file name for マップ名 (cr for none) ?
```

マップ名はシンボルファイルのファイル名です．ソース行を表示するためには，対応するソースファイルの名前を入力しなければなりません．ファイル名には，ファイル名拡張子が必要です．SYMDEBが指名されたファイルを見つけられない場合は，新しい名前を催促します．

ソース行の表示を中止したい場合は，SYMDEBがファイル名のプロンプトを表時した時点でリターンキーを押してください．SYMDEBは実際のソース行を中止して，代わりにマップ名と行番号を表示します．

例　Set Source Modeコマンドの使用例をつぎに示します．

```
S&
```

この例では，SYMDEBがソース行表示形式にセットされます．以後のコマンドに対して，SYMDEBは命令のソース行のみを表示します．

```
S&
```

この例では，SYMDEBがソース行と命令コードの表示形式にセットされます．以後のコマンドに対して，SYMDEBは，ソース行と命令コードの両方を表示します．

Shell Escape

書 式 !<MS-DOS コマンド>

機 能 **SYMDEB のモードからの，MS-DOS コマンドの実行．**

解 説 Shell Escape コマンド (!) は，SYMDEB のモードから，直接 MS-DOS のコマンドを実行します．このコマンドを用いると，デバッグ中でも (SYMDEB を中止せずに)，ディレクトリを調べたりすることができます．SYMDEB は，指定された <MS-DOS コマンド> (内部・外部コマンド共に) を実行し，コマンドが終了すると，再び SYMDEB のコマンド入力モードに戻ります．

MS-DOS コマンドの1つとして，MS-DOS のコマンドプロセッサ COMMAND.COM 自体を指定することができます (この場合，引数はつきません)．SYMDEB は，メモリへ COMMAND.COM をロードし，制御をこれへ移します．このときユーザーは，通常の MS-DOS モードと同じ操作を行うことができます．この状態から，SYMDEB のモードに戻るときは，"EXIT" コマンドを用います．

参考

Shell Escape コマンド (!) を実行するためには，デバッグ中のプログラムが不必要なメモリを解放するようになっていなくてはいけません．MS-DOS コマンドを実行するためのメモリが不足すると，SYMDEB は，そのむねを知らせるメッセージを表示します．

不必要なメモリを解放するためには，プログラムで MS-DOS のファンクションコール 4AH (割り当てられたメモリブロックの変更) を用います．このファンクションコールによって，MS-DOS は，COMMAND.COM をロードするメモリ領域を確保することができます．同様のことが，LINK (リンカ) の／CPARMAXALLOC スイッチでも行えます．詳しくは，"MS-DOS プログラマーズリファレンスマニュアル" を参照してください．

Microsoft C Ver 3.0 以上で開発したプログラムで，__main 関数の実行後であれば，自動的にメモリは解放されます．Microsoft Pascal, FORTRAN Ver 3.30 以上で開発されたプログラムでも，最初のプロシージャ (手続き) を実行後であれば，メモリは自動的に解放されます．SYMDEB 自身をロードした場合でも，メモリは解放されます．しかし，MASM または前述のコンパイラと同様の機能を持たないコンパイラで開発したプログラムのデバッグを行うときは，メモリの解放を行わないと，Shell Escaspe コマンド (!) は実行できません．

例　!（Shell Escape）コマンドの使用例をつぎに示します.

```
-! dir b: *.asm
```

この例は，ドライブB:のファイルで，ファイル名拡張子".ASM"のつくもののディレクトリ情報を表示します. MS-DOSモードで，同様のコマンドを実行したときと同じ表示がされたのち，再びSYMDEBのコマンド入力モードに戻ります.

```
! chkdsk b:
```

この例は，MS-DOSの外部コマンド"CHKDSK"を実行するものです.！(Shell Escape) コマンドでは，このように外部コマンド（ユーザーが開発したものも含む）も実行できます.

Source Line

書　式　．（ピリオド）

機　能　**現在のソースラインの表示**

解　説　Source Line コマンド（.）は，高級言語によるプログラムのデバッグに用いるコマンドです．たとえば，プログラムをブレークポイントで停止させた時点で，ピリオド（.）を入力すると，そのときのソースモード（Set Source　Mode コマンド参照）に関係なく，現在のソースラインを表示します．

なお，このコマンドは，MASM, SYMDEB, SYMDEB と互換性の無いコンパイラで開発したプログラムでは無効です．

例　.（Source Line）コマンドの使用例をつぎに示します．

```
-.
for  ( i  =  0;  i  <=  SIZE;  i++)   ;
-
```

この例は，SYMDEB と互換性のある，C コンパイラで開発したプログラムをデバッグ中のものです．

Stack Trace

書式　K [〈値〉]

機能　**スタックフレームの内容の表示**

解説　Stack Traceコマンド（K）は，現在のスタックフレームの内容を表示します．表示される項目は，現在のプロシージャと，それからコールされているプロシージャおよび引数です．さらに，各プロシージャをコールしている，ファイルとソースラインの行番号も表示されます．

パラメータ〈値〉が指定されると，その値の数まで引数が，16進数で表示されます．

例　Kコマンドの使用例をつぎに示します．

これは，デバッグ中のあるプログラムを実行し，ブレークポイントに達したところで，〈値〉を指定せずに，Kコマンドを入力した場合の表示例です．

```
-K
IGROUP:__max(0002,0003) form.sample.c:7
IGROUP:__main(?)
-
```

この例では，現在のプロシージャは〝__max〟で，2つの引数〝2H〟と〝3H〟を取っていることが分かります．また，このプロシージャ〝__max〟は，ソースファイル〝sample.c〟の7行目のプロシージャ〝__main〟からコールされており，〝__main〟の引数は不定であることも分かります．

〈値〉を指定した場合の出力はつぎのようになります．

```
-K 3
IGROUP:max (0002, 0003)    form..sample.c:7
IGROUP:main (0002, 2F62, 2EBA)
-
```

この例では，プロシージャ〝__main〟の引数が，3個まで表示されています．

Symbol Set

書　式　Z <シンボル> <値>

機　能　**シンボリックアドレスに値をセットする**

解　説　Symbol Set コマンドは，指定したシンボルのアドレスへ，値を入れます．

例　Z コマンドの使用例をつぎに示します．

```
-Z  close  26
```

この例は，シンボル〝close〞のアドレスへ，26 h をセットするものです．

Trace

書式　T [=〈開始アドレス〉] [〈カウンタ〉]

機能　プログラムの実行とトレース

解説　Trace コマンドは，入力された開始アドレスの命令を実行して，すべてのレジスタとフラグの現在値を表示します．表示の形式は Register コマンドと同じです．

〈開始アドレス〉を入力すると，コマンドは，入力されたアドレスの命令を開始します．入力しない場合は，コマンドは現在の CS レジスタと IP レジスタが示す命令の実行を開始します．開始アドレスを入力する場合は，イコール記号（=）が必要となります．

〈カウンタ〉を入力すると，コマンドは，その数の命令を実行するまで停止しません．コマンドは，つぎの命令を実行する前に各命令のレジスタとフラグの現在値を表示します．

参考

Trace コマンドは，8086，8088，186 または 286 マイクロプロセッサのハードウェアトレースモードを使用するので，ROM（読み取り専用メモリ）に記憶されている命令をトレースすることもできます．

例　T コマンドの使用例をつぎに示します．

```
T =_main
```

この例では，_main からの命令が実行され，レジスタとフラグの現在値が表示されます．また，次に実行する命令も表示されます．

```
T
```

この例では，現在の CS レジスタの IP レジスタが示す命令が実行されます．

```
T = 011A  10
```

この例では，現在の CS セグメントの 011A から，16（10H）個の命令の実行が開始されます．コマンドは，現在のレジスタとフラグの値を 16 回表示します．また，トレース中の命令の表示も行います．

Unassemble

書　式　U　[〈レンジ〉]

機　能　メモリ内容の逆アセンブル

解　説　Unassembleコマンドは，デバッグ中のプログラムの命令（インストラクション）とプログラム文，またはそのどちらかを表示します．表示の形式は，Set Source　Modeコマンドによってセットされている表示形式によって違います．表示形式には，つぎに示すものがあります．

形式：　逆アセンブルモード（S－）

機能：　命令コードの表示．SYMDEBは，範囲によって与えられるアドレスからメモリバイトを読み取り，このバイトをアセンブリ言語文に翻訳します．その結果の文の形式は，Assembleコマンドで定義するものと同じです．シンボルマップがプログラムと共にロードされている場合，ラベルや変数や絶対シンボルを表すオペランドは，アドレスではなく名前で表示されます．

形式：　ソースモード（S＋）

機能：　プログラムのソースファイル行の表示．SYMDEBは，入力された範囲の命令に対応するソース行を表示します．

形式：　ミックスモード（S&）

機能：　ソース行と命令コードの表示．SYMDEBは，アセンブリ言語文の対応する各グループに対して，1行のソース行を表示します．ソース行は，ソースファイルから読み取られます．アセンブリ言語文は，メモリバイトから翻訳されます．

SYMDEBでは，ソースモードの場合でも，ミックスモードの場合でも，シンボルマップがプログラムと共にロードされることと，ソースファイルの行番号情報がそのマップ内にあることが必要です．プログラムの入力された部分に対する行番号情報がない場合，SYMDEBは表示を行いません．

任意選択の〈レンジ〉を入力すると，コマンドは，入力されたレンジ内のコードから作られた命令を表示します．レンジを入力しないと，コマンドは，現在のアンアセンブルアドレスのコードの最初の8行から作られた命令を表示します．現在のアンアセンブルアドレスとは，前回のUnassembleコマンドが表示した最

終バイト（行）の次のバイト（行）のアドレスのことです．

逆アセンブルモードとミックスモードを用いる場合，SYMDEB は，8 ビットの即時オペランドの 16 進値と ASCII 値の両方を表示します．16 進値は命令の一部として表示され，ASCII 値は同じ表示行中のセミコロン（;）の直後に表示されます．

8086 ニーモニックコードと，8087 ニーモニックコードの表示のみが可能です．

例　U コマンドの使用例をつぎに示します．

●逆アセンブルモード

```
U  CS:02AD
```

この例では，アドレス “CS：02 AD” から 8 行目までの分解コードが表示されます．表示は，つぎのようになります．

```
1156:02AD  55          PUSH   BP
1156:02AE  8BEC        MOV    BP, SP
1156:02B0  B802C0      MOV    AX, 0002
1156:02B3  E893FF      CALL   chkstk
1156:02B6  C746FE6100  MOV    Word Ptr  [BP-02] ,0061
1156:02BB  FF0EEC05    DEC    Word Ptr  [05EC]
1156:02BF  833EEC0500  CMP    Word Ptr  [05EC] ,+ 00
1156:02C4  7C11        JL     02D7
```

```
U  __main    L  0A
```

この例では，アドレス “__main” の 10（16 進の 0 A）行の命令コードが表示されます．表示はつぎのようになります．

```
IGROUP: __main:
1156:02AD  55          PUSH  BP
1156:02AE  8BEC        MOV   BP, SP
1156:02B0  B802C0      MOV   AX, 0002
1156:02B3  E893FF      CALL  chkstk
1156:02B6  C746FE6100  MOV   Word Ptr [BP-02] ,0061
1156:02BB  FF0EEC05    DEC   Word Ptr [05EC]
1156:02BF  833EEC0500  CMP   WordPtr [05EC] ,+ 00
1156:02C4  7C11        JL    __main + 2A(02D7)
1156:02C6  8AA6FE      MOV   AL, [BP-02]
1156:02C9  8B1EEA05    MOV   BX, [05EA]
```

●ソースモード

```
U  CS:02AD
```

この例では，プログラムソースファイルから8行のソース行が表示されます．これらの行は，アドレス〝CS：02 AD〞から始まる命令コードに対応します．表示は，つぎのようになります．

```
4:{
5:    int i;
6:
7:    for(i = 'a'; i < 'z'; i ++)
8:    putchar(1);
9:    for(i = 'A'; i < 'z'; i ++)
10:   putchar(1);
11:   for(1 = '0'; 1 < 'g'; 1 ++)
```

```
U  __main  L  5
```

この例では，アドレス〝__main〞から5行目までのソース行が表示されます．表示はつぎのようになります．

```
4:{
5:    int i;
6:
7:    for(i = 'a'; i < 'z'; i ++)
8:          putchar(1);
```

●ミックスモード

```
U  CS:02AD
```

この例では，"CS:02AD" から 8 行目までの命令コードとプログラムソースコードが表示されます．表示は，つぎのようになります．

```
4:{
IGROUP:_main:
1156:02AD 55            PUSH   BP
1156:02AE 8BEC          MOV    BP,SP
1156:02B0 B80200        MOV    AX,0002
1156:02B3 E893FF        CALL   chkstk
7:          for(1 = 'a'; 1 < 'z'; 1 ++)
1156:02B6 C746FE6100  MOV    Word Ptr [BP-02],0061
```

```
U  main  L  5
```

この例では，アドレス "_main" の 5 行の命令コードとプログラムソースが表示されます．表示は，つぎのようになります．

```
4:{
IGROUP:_main:
1156:02AD 55            PUSH   BP
1156:02AE 8BEC          MOV    BP,SP
1156:02B0 B80200        MOV    AX,0002
```

View

書　式　V.[<アドレス>]

機　能　**指定したアドレスからソースラインを表示**

解　説　Viewコマンドは，高級言語によるプログラムに用いるコマンドで，そのときのソースモードの設定（Set Source Modeコマンド参照）に関係なく，指定したアドレスから始まるソースラインを表示します．

このコマンドを用いるためには，ソースプログラムファイル中のソースラインの行番号に関する情報を持ったシンボルファイルが必要で，MASM, SYMDEB, SYMDEBと互換性の無いコンパイラで開発したプログラムのデバッグでは，このコマンドは無効です．

例　Vコマンドの使用例をつぎに示します．

```
-V_func
4:{
5:    int  i;
6:
7:    for   (i = 'a'; i < 'z'; i++)
8:    putchar  (i)  ;
9:    for   (i = 'A'; i < 'Z'; i++)
10:   putchar  (i)  ;
11:   for   (i = '0'; i < '9'; i++)
-
```

この例は，Cコンパイラによって開発したプログラムをデバック中の様子で，アドレス〝_func″から始まるソースラインを表示したものです．同様のことが，FORTRANやPascalで開発したプログラムでも行うことができます．

Write

書 式　W [〈アドレス〉[〈ドライブ〉 〈レコード〉 〈カウント〉]]

機 能　**ファイルまたはメモリ内容のディスクへの書き出し**

解 説　Writeコマンドは，入力された記憶位置の内容を指名されたファイルやディスク上の入力された論理レコードへ書き出します．

ファイルへ書き出す場合は，Nameコマンドを用いてファイル名をセットし，次にRegisterコマンドを用いて，BX:CXレジスタのペアを書き出すべきバイト数にセットしなくてはなりません．〈アドレス〉を入力しないと，コマンドは，CSレジスタの現在値であるアドレス〝CS:100〟から始まるバイトをコピーします．〈アドレス〉を入力すると，コマンドは，そのアドレスから始まるバイトをコピーします．

注意

Goコマンド，PTraceコマンド，Traceコマンドが実行済の場合，ロードされたファイルのファイル名，長さおよび開始アドレスを正しい値にセットしなくてはなりません．Registerコマンドを用いてBXレジスタとCXレジスタを修正してある場合，これらのレジスタも正しい値にセットしなくてはなりません．

ディスク上の論理レコードへ書き出す場合は，〈アドレス〉，〈ドライブ〉，〈レコード〉および〈カウント〉を入力する必要があります．〈ドライブ〉は，書き出すべきドライブを指名する番号です（通常，各々ドライブA:(0)，B:(1)，C:(2)，D:(3)を表します)．〈レコード〉は，データを受け取るべき最初の論理レコードを指定する1文字から4文字の16進数です．〈カウント〉は，ディスクへ書き出すべきレコード数を指定します．

警告

ディスクセクタが空いていることを確かめずに，そこへデータを書き込んではいけません．予約セクタや専有セクタへ書き出すと，ファイルの内容，さらにはディスク全体が破壊されることがあります．

例　　Wコマンドの使用例をつぎに示します．

```
N  table.bin
R  BX
BX  0100
:  0000
R  CX
CX  D43C
:  0100
W  DGROUP:table
```

この例では，256（100H）バイトがディスク上の〝table.bin〞という名前のファイルへ書き出されます．書き出すべきバイトは，アドレス〝DGROUP:table〞から始まります．

```
W  workspace  2  34  3
```

この例では，レコード番号52（34H）から3つ目までの論理レコードがドライブCへ書き出されます．書き出されるバイトは，アドレス〝workspace〞から始まります．

3.6 エラーメッセージ

SYMDEBは，実行できないコマンドを見つけると，エラーメッセージを表示します．SYMDEBは，起きたエラーに続けて“Error”というメッセージを表示します．記号(へ)は，コマンド行中のエラーの正確な位置を示すものです．たとえば，つぎに示す表示は，Dumpコマンドの値の範囲が適切でないことを示しています．

```
d  cs:100  0
           ^ Error
```

また，SYMDEBは，他のエラーメッセージを表示して，そのエラーの内容をくわしく伝える場合もあります．表示されるエラーメッセージをつぎに示します．エラーによってそのエラーに関連したSYMDEBコマンドは終了しますが，SYMDEB自体が終了することはありません．

Bad Flag!

フラグが間違っています．ユーザーはフラグを変更しようとしましたが，タイプした文字は許容されているフラグ値ではありませんでした．正しいフラグ入力値のリストについては，Registerコマンドを参照してください．

Too many breakpoints!

Goコマンドのパラメータとして与えられたブレークポイントの数が10個を越えています．10個以下のブレークポイントでGoコマンドを入力し直してください．

Bad register

Registerコマンドで，間違ったレジスタ名が入力されました．正しいレジスタ名のリストについては，Registerコマンドを参照してください．

Double flag

1つのフラグに対して2つの値を入力しました．指定できるフラグ値は1つだけです．Registerコマンドを参照してください．

Breakpoint list or ' * ' expected!

Breakpoint Clearコマンド，Breakpoint Disableコマンド，またはBreakpoint Enableコマンドで，ブレークポイントのリストを入力せずにコマンドを実行しました．

Error reading .SYM file

SYMDEBコマンド行に指定したシンボルファイル名が，読み取れませんでした．この場合，ファイルの長さが0になるか，ディスクエラーが起きます．

付録　SYMDEBの使用できるアセンブラとコンパイラ

SYMDEB／MAPSYMのシンボリックデバッグ機能はつぎに示す言語で記述されたプログラムに使用できます．

microsoft FORTRAN version 3.0 以上
microsoft Pascal version 3.0 以上
microsoft C version 2.0 以上
microsoft Macro Assembler version 1.0 以上
microsoft Basic Compiler version 1.0 以上
microsoft Business Basic Compiler version 1.0 以上

SYMDEB／MAPSYMのソース行表示機能は，正しい行番号の情報を生成するコンパイラで使用できます．ソース行番号の情報を自動的に生成するコンパイラをつぎに示します．

microsoft FORTRAN version 3.0 以上
microsoft Pascal version 3.0 以上

つぎに，ソース行番号の情報を生成する場合にコマンド行のスイッチが必要なコンパイラを示します．正しいコンパイラのスイッチについては，各言語のマニュアルを参照してください．

microsoft C version 2.0 以上

第4章

LIB：ライブラリマネージャ

4.1 イントロダクション

マイクロソフトライブラリマネージャ；LIB は，ライブラリファイルの作成，管理を行うためのユーティリティです．

ライブラリファイルは，他のプログラムが実行時に必要とする，1つ以上の〝オブジェクトモジュール〟の集まり（コレクション）です．オブジェクトモジュールは，データやインストラクションがアセンブルやコンパイルされたもので，他のプログラムとリンクできるようになっています．

ライブラリは，リンカ；LINK によって用いられます．LINK は，プログラム中で定義されていないルーチン（外部参照）を解析し，それをライブラリファイルから捜し出して，参照を解決し，実行可能なプログラムを作成します．

LIB は，1つ以上の〝オブジェクトファイル〟のコピーを取り込んで，ライブラリファイルを作成します．オブジェクトファイルは，1つのオブジェクトモジュールから成るもので，MASM や高級言語コンパイラで作成します．

LIB は，オブジェクトモジュールをライブラリに追加・登録すると，このモジュール名をライブラリファイルの〝ライブラリ見出しテーブル〟に登録します．LINK は，ライブラリファイルを参照するとき，先ずこの見出しテーブルをチェックし，必要なモジュールが登録されているか調べます．目的のモジュールが見つかると，LINK はこれを処理中のプログラムへコピーしリンクします．

本章では，ライブラリファイルの作成方法，LINK での処理効率を向上させる方法，モジュールの追加・削除・入れ換え方法などを解説します．

4.2 LIBの起動と使用方法

本節では，LIBの起動方法，ライブラリファイルの作成・編集方法を解説します．

LIBの使用方法は，3通りあります．

1. コマンドラインによる方法．MS-DOSからLIBを起動するコマンドライン上に，ライブラリの編集の指示を含めてしまう方法です．
2. コマンドプロンプトによる方法．LIBを起動したときに表示されるコマンドプロンプトに答えながら，編集指示を行う方法です．
3. 応答ファイルによる方法．編集指示だけを収めた〝応答ファイル〟を作っておき，このファイルに従った編集をするように，LIBへ指示する方法です．

LIBは，ユーザーの指示に従った処理を行うとともに，指示が不足しているときは，必要な情報の入力を求めるコマンドプロンプトを表示します．

なお，LIBによる処理を中止するときは，CTRL＋Cキーを押します．

4.2.1 コマンドラインによる方法

LIBを起動するための，MS-DOSのコマンドライン上で，ライブラリの編集に関する指示（パラメータ）も指定する方法です．そのための，コマンドラインの書式はつぎのとおりです．

LIB　〈ライブラリ名〉[／PAGESIZE：n] コマンド・・・
[，リストファイル名，出力ファイル名]

〈ライブラリ名〉は，処理の対象とするライブラリファイル名です．新規に作成する，編集を加えるライブラリファイル名を指定します．なお，ライブラリファイル名の拡張子が〝．LIB〟の場合は，これを省略することができます．

／PAGESIZE：nは，必要に応じて指定するスイッチです．これは，ライブラリのページサイズ（6．2．6．ページサイズの設定参照）を指定するもので，指定を省略した場合は，自動的にページサイズは16となります．

〈コマンド〉は，LIBに対する指示です．ライブラリファイルに追加，削除などを行うモジュール名を指定します．LIBのコマンドについては，〝6．3章　LIBのコマンド〟を参照してください．

〈リストファイル名〉は，ライブラリのクロスリファレンスリストファイルを作成したいときに指示します．このクロスリファレンスリストファイルには，ライブラリ中のモジュールのPUBLICシンボル名が，一覧表になって収められます．なお，リストファイル名の指定はオプションで，省略するとこのファイルは生成されません．

〈出力ファイル名〉は，LIBで処理を行った結果を収める新しいライブラリファイル名です．

通常(出力ファイル名を指定しなかった場合)，LIBは指定したライブラリファイルそのものを編集しますが，この出力ファイル名を指定すると，編集の対象になったライブラリは元の状態で残され，編集（コマンド）が反映された新しいライブラリファイルが作られます．この〈出力ファイル名〉の指定はオプションで，古い（元にする）ライブラリファイルをそのままの状態で残しておきたいときに指定します．

ライブラリファイルがカレントディレクトリに無い場合は，ドライブ名やパス名を含めてファイル名を指定してください．

リストファイル名を指定するときは，最後の〈コマンド〉との間に，区切り記号のカンマ(,)をつけます．この区切り記号は，リストファイル名と出力ファイル名の間にも必要です．なお，リストファイル名を省略して出力ファイル名を指定するときは，〈コマンド〉との間にカンマを2つ（,,）入れます．

パラメータの後ろにセミコロン（;）をつけると，それ以降のパラメータはLIBの初期設定に従って処理されます．セミコロンはコマンドラインの最後につき，セミコロンの後ろにパラメータをつけても無効となります．

例：

コマンドラインにパラメータをつけて，LIBを起動する例をつぎにしめします．

```
LIB  lang  −+heap;
```

この例で，処理の対象となるライブラリファイルは〝lang.lib〟です．このライブラリファイルの拡張子は〝.LIB〟なので，これを省略して指定しています．

ライブラリに加える編集(コマンド)は，〝−+heap〟で，ライブラリ中のモジュール〝heap〟を新しいモジュールに交換します．LIBは，このコマンドに従って，ライブラリ中のモジュール〝heap〟をオブジェクトファイル〝heap.obj〟の内容と置き換えます．

コマンドラインの最後にセミコロン（;）がついているので，〈コマンド〉よりも後ろのパラメータに関しては，初期設定に従った処理が行われます．すなわち，リストファイルは生成されず，編集はライブラリファイルに直接反映され，新しいライブラリ（出力ファイル）は作成されません．

つぎの例は，パラメータを一通り指定した場合のものです．

```
LIB  lang  −+heap,lang.lst,lang1.lib
```

この例は，前例と同様に，ライブラリ〝lang.lib〟のモジュール〝heap〟を置き換えるものですが，リストファイル名と出力ファイル名が指定されています．LIBは，クロスリファレンスリストファイル〝lang.lst〟を作成します．また，編集結果は，出力ファイル〝langl.lib〟に収められ，編集の元になったライブラリ〝lang.lib〟は，元の状態のまま残されます．

4.2.2　コマンドプロンプトによる方法

ライブラリマネージャの名前〝LIB〞だけを入力すると，LIB は，処理に必要な情報の入力を求めるコマンドプロンプトを表示します．コマンドプロンプトに沿った入力方法は，つぎのとおりです．

1. MS-DOS のプロンプト（A>，B>，･･･）が表示されていることを確かめ，つぎのようにタイプして，リターンキーを押します．

   LIB

   LIB が起動し，ライブラリ名の入力を求める，第1のコマンドプロンプトが表示されます．

   Library name:

2. 作成・編集したい，ライブラリファイル名をタイプします．ライブラリファイル名の拡張子が〝.LIB〞のときは，これを省略することができます．新しいライブラリファイルを作成する場合も，ここでそのファイル名を指定します．ファイル名をタイプしたら，リターンキーを押します．
   リターンキーが押されると，LIB は入力された名前のライブラリファイルを捜します．それが見つからないときは，新しい名前のライブラリファイルを作成するか否かの判断を求める，補助のプロンプトを表示します．これは，ファイル名の入力ミスをチェックする役割も兼ねています．

   Library file does not exist. Create?

   〝Yes〞をタイプすると，新しいライブラリファイルが作成されます．〝No〞をタイプすると，MS-DOS のコマンドレベルに戻ります．
   ライブラリファイル名の入力が済むと，LIB は，ライブラリに行う編集処理（オペレーション）の指示を求める第2のプロンプトを表示します．

   Operations:

3. ライブラリに行う編集操作（コマンド）をタイプして，リターンキーを押します．指示するコマンドが多く，入力行が不足するときは，その行の最後にアンパサンド記号（&）をタイプして，リターンキーを押します．LIB は，同じコマンドプロンプトを次の行に表示するので，コマンドの指示のつづきをタイプします．
   すべてのコマンドをタイプしたら，リターンキーを押します．ライブラリファイルの整合性をチェックするときは，コマンドをタイプせずに，リターンキーだけを押してくだ

さい．リターンキーが押されると，LIBは，リストファイル名の入力を求める，第3のコマンドプロンプトを表示します．

List file:

4. 新しいクロスリファレンスリストファイルの名前をタイプし，リターンキーを押します．クロスリファレンスリストファイル名は，ユーザーが自由につけることができます．LIBは，このファイル名の拡張子を自動的につけることはしません．クロスリファレンスリストファイルが不必要なときは，何もタイプせずに，リターンキーを押してください．ライブラリファイルが新規に作成される場合，表示されるコマンドプロンプトはここまでです．既にあるライブラリファイルを編集する場合は，つづいて出力ファイル名の入力を求める，最後のプロンプトが表示されます．

Output library:

5. ライブラリに編集を行った結果（出力）を収める，新しいライブラリファイル名（出力ファイル名）を入力します．ファイル名拡張子を省略すると，LIBは，自動的に〝.LIB″という拡張子をつけます．出力ファイル名を指定せずにリターンキーを押すと，LIBは，編集したライブラリファイル名を出力ファイル名に用います．この場合，編集の元になったライブラリファイルの拡張子が〝.BAK″につけかえられます．

出力ファイル名をタイプしたら，リターンキーを押してください．LIBは，ライブラリの作成・編集処理を開始します．

参 考

第2のコマンドプロンプト以降の入力で，セミコロン（；）を入力すると，それ以後の処理が，LIBの初期設定に従って行われます．LIBは，セミコロンが入力されると，それ以後のコマンドプロンプトの表示をせずに，ライブラリの編集を開始します．

処理に関係したファイルが，カレントディレクトリ以外にあるときは，ドライブ名，パス名を含めた正しいファイル名を指定してください．

ユーザーは，ライブラリファイルのページサイズを指定することができます．この指定は，ライブラリファイル名の指定と共に行います．詳しくは，4．2．6〝ライブラリページサイズの設定″を参照してください．

例：

LIBの表示するコマンドプロンプトに沿った，入力例をつぎに示します．

```
LIB
Library File: math
Operations: +sin +cos &
Operations: +atan +exp
List file:
Output library: math1
```

この例では，ライブラリファイルMATH.LIB″が処理の対象になっています．LIBの設定を利用して，ここでは拡張子を省略して，ファイル名を指定しています．〈コマンド〉では，4つのオブジェクトモジュール（sin, cos, atan, exp）の追加が指定されています．各モジュールは，SIN.OBJ, COS.OBJ, ATAN.OBJ, EXP.OBJというファイルに収められています．コマンドの入力を求める，2行目のコマンドプロンプトの最後にタイプしてあるアンパサンド記号（&）は，コマンド行を拡張する場合の例です．このように，プロンプト〝Operations:″の入力行の最後にアンパサンド（&）をタイプしてリターンキーを押すと，同じコマンドプロンプトが表示され，指定したいコマンドの続きを入力することができます．この例では，クロスリファレンスリストファイルの出力を求めていないので，このプロンプトに対しては何もタイプせずにリターンキーを押しています．ライブラリ〝MATH.LIB″に4つのオブジェクトモジュールを追加した結果（新しいライブラリ）は，〝MATH1.LIB″という名前のファイルに収められます．編集の元になったライブラリ〝MATH.LIB″は，そのままの内容で保存されます．

4.2.3　応答ファイルによる方法

ファイル名やコマンドなど，LIBに対する入力だけを収めた〝応答ファイル″を用意しておき，この応答ファイルの内容に従って処理を進める方法です．このときの入力は，つぎのようになります．

LIB　@〈応答ファイル名〉

応答ファイルを用いるときは，アットマーク（@）を先頭につけて，〈応答ファイル名〉を指定します．このアットマーク（@）は，後ろにつづくファイル名が（ライブラリファイル名ではなく）応答ファイルであることを表します．また，応答ファイルがカレントディレクトリ以外にあるときは，ドライブ名，パス名を含めて正しいファイル名を指定してください．

応答ファイルの名前は，自由につけることができます．このファイルの内容は，つぎのような形式で作ってください．

〈ライブラリファイル名〉［／PAGESIZE：n］
〈コマンド〉
〈クロスリファレンスリストファイル名〉
〈出力ファイル名〉

応答ファイル中の各行が，前節で解説した各コマンドプロンプトに対応しています．〈コマンド〉が多く，1行に収まらないときは，その行の最後にアンパサンド記号（&）をつけて，次の行にコマンドのつづきを書きます． 応答ファイルを利用する方法でLIBを起動すると，LIBは，コマンドプロンプトとそれに対応する応答ファイルの内容を表示しながら処理を進めます．応答ファイルの内容が不足しているとき，LIBは，該当するコマンドプロンプトを表示し，ユーザーの入力を待って処理を再開します．

応答ファイル中でも，LIBの初期設定に従った処理を指示する，セミコロン（；）を利用することができます．

例：

応答ファイルの例をつぎに示します．

```
PLIB
+cursor +heap -heap *stack
cross.lst
```

この例で，処理されるライブラリファイル名は〝PLIB.LIB〟です．コマンドは，モジュール〝cursor〟の追加，〝heap〟の置き換え，〝stack〟のコピーが指示されています．また，クロスリファレンスリストファイル〝cross.lst〟が，作成されます．

4.2.4 新しいライブラリの作成

新しいライブラリファイルを作成するときは，ライブラリファイル名の入力で，希望するファイル名を指定します．新しいライブラリの名前は，既にあるライブラリ名と重複しないようにつけてください．指定した名前のファイルが既に存在していると，LIBは，そのファイルが編集されるものと判断します．指定した名前のファイルが存在していない場合，LIBは，つぎのプロンプトを表示します．

```
Library file does not exist. Create?
```

〝Yes〞をタイプすると，新しいライブラリファイルが作成されます．〝No〞をタイプすると，LIBの処理が中止され，MS-DOSに戻ります．

新しく作成するライブラリファイルについても，ファイル名と共に，ページサイズを指定することができます．ページサイズの指定を省略すると，LIBの初期設定に従って，ページサイズは16バイトになります．詳しくは，4．2．6〝ページサイズの設定〞を参照してください．

4.2.5 ライブラリの整合性のチェック

ライブラリファイル名だけを指定して，コマンドを指定しないと，LIBは，ライブラリの整合性をチェックします．

ライブラリの整合性をチェックするときは，〝LIB〞につづいてライブラリファイル名指定し，セミコロン（；）をタイプして，リターンキーを押します．LIBは，ライブラリ中のすべてのモジュールがアクセスできるかどうか調べます．調べたライブラリ中になんらかの問題があった場合は，エラーメッセージを表示します．問題がなかったときは，特にメッセージは出力されず，MS-DOSのプロンプトが表示されます．

例：

整合性のチェックを行う場合の，入力例をつぎに示します．

```
LIB  math;
```

この例は，ライブラリファイル〝MATH.LIB〞のチェックを行うものです．

ライブラリの整合性のチェックを行う場合としては，ファイルのベリファイによって，ライブラリ見出しテーブルと実際の内容に差がないことを確かめるなどが考えられます．たとえば，ライブラリファイルを他のディスクからコピーした後など，この整合性のチェックを行って，コピーが正確に行われたか否かを確認するとよいでしょう．

なおLIBは，モジュールを追加するときは，事前に整合性のチェックを自動的に行うように設定されています．したがってユーザーは，モジュールを追加する度に整合性チェックを行う必要はありません．

4.2.6　ライブラリページサイズの設定

ユーザーはライブラリのページサイズを設定することができます．ページサイズの設定は，編集・作成するライブラリファイル名の指定と共に行います．コマンドラインの書式は，つぎのようになります．

LIB　〈ライブラリ名〉／PAGESIZE：n

ここで，nが希望するページサイズで，16～32768の間の2の乗数を指定することができます．

ページサイズの設定は，ライブラリ中のモジュールの配置（配列）を制御するものです．モジュールは，常に，ページサイズ（バイト単位）の倍数の位置からスタートされるように配置されます．ライブラリを新規に作成するときのLIBの初期設定や，既にあるライブラリのページサイズは，16（バイト）となっています．

参　考

ページサイズは，LIBの採用しているインデックス技法と関連しています．この方法では，ページサイズは大きいほど，ライブラリに多くのモジュールを登録することができます．しかし，ライブラリ中の各モジュールは，平均n／2バイト（nはページサイズ）の無駄なスペースを含んでいます．したがって，実用上は小さなページサイズの方が有益なことが多く，ユーザーは可能なかぎり小さなページサイズを指定した方がよいでしょう．

例：

ページサイズの設定例をつぎに示します．

LIB　math／PAGESAIZE：256

この例は，ライブラリ "MATH．LIB" のページサイズを256（バイト）に設定したものです．

4.2.7　クロスリファレンスリストの生成

LIBに対し，クロスリファレンスリストファイル名を指定すると，クロスリファレンスリストを得ることができます．このクロスリファレンスリストには，2種類のリストが収められます．第1は，ライブラリ中のすべてのパブリックシンボルのリストで，第2は，ライブラリ中のすべてのモジュールのリストです．

第1のリストには，すべてのシンボルがアルファベット順に収められます．各シンボルは，それを参照しているモジュール名の後に表示されます．この部分の例をつぎに示します．

```
START . . . . . . . main
SUM . . . . . . . . add
SUM2. . . . . . . . add
EXIT . . . . . . . . error
```

第2のリストには，すべてのモジュールがアルファベット順に収められます．モジュール名は，そのモジュール中で参照されるアルファベット順のパブリックシンボルの後ろに表示されます．この部分の例をつぎに示します．

```
main      Offset:    00000200 H     Code and data size: 20 H
       START
add       Offset:    00000400 H     Code and data size: 20 H
       SUM                    SUM 2
error     Offset:    00000600 H     Code and data size: CH
       EXIT
```

4.3　LIBのコマンド

〈コマンド〉の指示に用いる，LIBのコマンドにはつぎのようなものがあります．モジュールの追加・削除・置換はライブラリの編集に用い，コピー・移動は新しいライブラリの作成に関係したコマンドです．

コマンド	記　号
追加	＋
削除	－
置換	－＋
コピー	*
移動	－*

これらのコマンドは，処理の対象となるオブジェクトファイル名・オブジェクトモジュール名の先頭につけて使用します．

4.3.1 モジュールの追加

書 式

+<オブジェクトファイル名>

追加（+）コマンドは，オブジェクトモジュールをライブラリに登録するときに用います．<オブジェクトファイル名>は，追加したいモジュールを収めたファイル名です．このファイルの拡張子が"．OBJ"のときは，これを省略することができます．ファイルがカレントディレクトリ以外に存在しているときは，ドライブ名やパス名などを含めた，ただしいファイル名を指定してください．追加コマンドを示す記号(+)と，オブジェクトファイル名の間には，スペース（空白）を入れないでください．

LIBは，指定されたオブジェクトファイルを捜し，これの内容ををライブラリに追加します．つづいて，オブジェクトファイル名からドライブ名，パス名，ファイル名拡張子を取り除き，残った部分の名前をライブラリ見出しテーブルに登録します．

なお，モジュールは，常にファイルの後ろへ追加されます．

例：

ライブラリへ，モジュールを追加する例をつぎに示します．

```
LIB math +sin.obj
```

この例は，オブジェクトファイル"sin.obj"中のモジュールをライブラリ"math.lib"に追加するものです．

```
LIB ¥lib¥math +cos, list;
```

この例は，¥libディレクトリのライブラリ"math.lib"へ，オブジェクトファイル"cos.obj"中のモジュールを追加するものです．さらに，クロスリファレンスリストファイル"list"の作成が指定されています．出力ファイルに関しては，LIBの初期設定が用いられます．

```
LIB math +A:¥src¥atan;
```

この例は，カレントディレクトリのライブラリ"math.lib"へ，ドライブA：のサブディレクトリ"src"のオブジェクトファイル"atan.obj"中のモジュールを追加するものです．リストファイル，出力ファイルに関しては，LIBの初期設定が用いられます．

4.3.2　モジュールの削除

書　式

－〈モジュール名〉

削除コマンド（－）は，〈モジュール名〉で指定されたオブジェクトモジュールを指定されたライブラリファイルから削除します．〈モジュール名〉は，ライブラリ見出しテーブルに登録されている名前で，正確に指定してください．

参　考

LIBは，コマンドライン上の順番に関係なく，モジュールの追加に先立って，削除を行います．したがって，モジュールの更新（新バージョンへの変更）は，確実に行われます．

例：

削除コマンドの使用例をつぎに示します．

```
LIB  math  －sin
```

この例は，ライブラリファイル〝math.lib〞の，モジュール〝sin〞を削除するものです．

```
LIB  ¥lib¥math  －cos, list;
```

この例は，サブディレクトリ〝lib〞の，ライブラリファイル〝math.lib〞から，モジュール〝cos〞を削除するものです．さらに，編集の終わったライブラリファイルの，クロスリファレンスリストファイル〝list〞を作成します．

```
LIB  math  ＋a:¥src¥atan  －atan;
```

この例は，ライブラリファイル〝math.lib〞の，モジュール〝atan〞を削除し，ドライブA：のディレクトリ〝src〞にあるオブジェクトファイル〝atan〞を追加します．ここでは，同じ名前のモジュールの追加が先に指定されていますが，LIBは，コマンドライン上で指定される順番に関係なく，削除を先に行います．

4.3.3　モジュールの置換

書　式

－＋〈モジュール名〉

置換コマンド（－＋）は，指定されたモジュールをライブラリファイルから削除するとともに，同じ名前のオブジェクトモジュールファイルをライブラリに追加します．指定する〈モジュール名〉は，ライブラリ見出しテーブルに登録されている名前で，正確に指定してください．

LIB は，まず指定されたオブジェクトモジュールをライブラリから削除します．つぎに，カレントディレクトリ中で，削除した〈モジュール名〉に拡張子〝.OBJ″のついたファイルを捜し，これを追加します．もし，該当する名前のオブジェクトモジュールファイルが見つからないときは，エラーメッセージが表示されます．

例：

置換コマンドの使用例をつぎに示します．

```
LIB math －＋cos ;
```

この例は，ライブラリファイル〝math.lib″の，モジュール〝cos″を削除し，オブジェクトモジュールファイル〝cos.obj″を追加します．

4.3.4　モジュールのコピー

書　式

＊〈モジュール名〉

コピーコマンド（＊）は，指定された名前のモジュールをコピーし，同じ名前のオブジェクトモジュールファイルを作成します．〈モジュール名〉は，ライブラリ見出しテーブルに登録されている名前で，正確に指定してください．

コピーしたモジュールが収められるオブジェクトファイル名は，〈モジュール名〉と同じ名前で，拡張子〝.OBJ″がつけられます．また，このファイルは，カレントディレクトリに作成されます．

例：

コピーコマンドの使用例をつぎに示します．

```
LIB math * cos;
```

この例は，ライブラリファイル〝math.lib〞の，モジュール〝cos〞をコピーして，オブジェクトモジュールファイル〝cos.obj〞を作成します．

4.3.5　モジュールの移動

書　式

　　－＊<モジュール名>

移動コマンド（－＊）コマンドは，指定されたモジュールをライブラリファイルから抜き取り，オブジェクトモジュールファイルに収めます．<モジュール名>は，ライブラリ見出しテーブルに登録されている名前で，正確に指定してください．

抜き取ったモジュールが収められるオブジェクトファイル名は，＜モジュール名＞と同じ名前で，拡張子〝.OBJ〞がつけられます．また，このファイルは，カレントディレクトリに作成されます．また，オブジェクトモジュールファルに移されたモジュールは，ライブラリから削除されます．

例：

移動コマンドの使用例をつぎに示します．

```
LIB math －＊cos
```

この例は，ライブラリファイル〝math.lib〞の，モジュール〝cos〞を抜き取り，オブジェクトモジュールファイル〝cos.obj〞に移します．モジュール〝cos〞は，ライブラリから削除されます．

4.3.6 ライブラリの連結

書 式

+<ライブラリ名>

追加コマンド（+）を用いると，編集を指定したライブラリへ，他のライブラリファイル全体を追加することもできます．指定する<ライブラリ名>が，追加するライブラリファイルで，追加するライブラリファイル<ライブラリ名>の，ファイル名拡張子が〝.LIB″のときは，これを省略することができます．

追加されるライブラリは，編集を指定したライブラリの後ろへ，既にあるモジュールを変更・削除することなく追加されます．

参 考

LIBは，XENIX方式，インテル方式のライブラリをMS-DOSのライブラリへ追加することもできます．

例：

ライブラリ単位の，モジュールの追加例をつぎに示します．

```
LIB  math1  +math.lib;
```

この例は，編集中のライブラリファイル〝mathl.lib″へ，ライブラリファイル〝math.lib″を追加するものです．

第5章
MAKE：プログラムメインテナ

5.1 イントロダクション

プログラムメインテナ：MAKEは，マクロアセンブラや高級言語によるプログラムの開発工程の保守・管理を自動的に行うユーティリティです．MAKEは，ソースファイルの更新に伴なって，それから派生するプログラムの更新など必要な処理をすべて自動的に実行します．

MAKEの機能は，いわゆるバッチ処理とは異なり，さらに高度です．メインテナンスの目的とするファイル(オブジェクトファイル，ライブラリファイル，クロスリファレンスリストファイルなど）と，それの元になる関連ファイル（ソースファイル，クロスリファレンスファイル，オブジェクトファイルなど）の作成日付を比較し，目的ファイルの日付が遅れている場合にだけ，特定の更新に必要な処理を実行します．MAKEは，さらにMAKEによる処理で生じたファイルの作成日付の食い違い（日付の遅れ）に関する処理も行います．

MAKEはバッチ処理のように，ファイルが1つでも更新されれば，すべてのファイルのアセンブル，コンパイル，リンクを行うようなことはしません．MAKEを使用することによって，多くのソースファイルを持つプログラムや，完成までに多くのステップを必要とするプログラムの作成で，かなりの手間と時間が節約されます．

つぎの項では，MAKEの使用方法を説明し，例を挙げてアセンブリ言語のプログラムの保守方法を示します．

5.1.1 MAKEの使用

MAKEを使用する場合は，実行したい作業を定義する〝メイクファイル〟と，この作業が依存するファイル〝関連ファイル〟をあらかじめ作成しておきます．

メイクファイルがあれば，MAKEを起動する場合，ただメイクファイル名を入力するだけで，自動的にファイルの内容を読み取られ，必要な処理を実行します．

本節では，メイクファイルの作成方法と，MAKEの起動方法を説明します．

5.1.2　メイクファイルの作成

メイクファイルは，テキストエディタを用いて作成します．ファイルは1つまたは複数の〝作業記述〟から成ります．記述の形式をつぎに示します．

〈目的ファイル名〉：〈関連ファイル名〉・・・
〈コマンド〉

目的ファイル名は，更新を必要とするファイルの名前です．関連ファイル名は，目的ファイルが依存する（作業に必要となる）ファイルの名前です．コマンドは，MS-DOSの外部コマンドです．

目的ファイルと関連ファイルは，有効なファイル名でなければいけません．これらのファイルがメイクファイルと異なるディレクトリにある場合は，パス名（ディレクトリ名）を含めた正しいファイル名を指定してください．また，目的ファイル名にドライブの指定はできません．

関連ファイルはいくつ入力してもよいが，目的名は1つしか許されません．関連ファイル名は，1つ以上のスペースで区切らなくてはなりません．関連ファイルが1行にはいりきらない場合，行の終りに円記号（¥）をタイプすれば，次の行に名前を続けることができます．

〈コマンド〉は，MS-DOSの外部コマンドで，TYPEやCOPYなどの内部コマンドではありません．コマンドは，複数指定することができますが，各々は必ず新しい行から始め，先頭にタブかスペースをいれなくてはなりません．コマンドは，目的ファイルの作成日付以後，関連ファイルの修正があった場合にだけ実行されます．

メイクファイルの中には，作業記述をいくついれてもかまいませんが，ある記述の最終行と次の記述の第1行との間には，必ず最低1行空けて区切らなくてはなりません．

注意

作業記述を書く順番は重要です．MAKEは，順に記述を調べ，ファイルの作成日付に基づいて，指定された作業を実行するかどうかを決定します．後のコマンドが目的ファイルを修正する場合，そのファイルを目的ファイルとする前の作業記述へ戻ることはできません．

例：

メイクファイルの例をつぎに示します．

```
startup.obj:        startup.asm
        MASM startup, startup, nul, nul

print.obj:          print.asm
        MASM print, print, print, print

print.ref:          print.crf
        CREF print, print

print.exe:          startup.obj  print.obj  ¥lib¥syscal.lib
        LINK startup+print, print, print/map, ¥lib¥syscal;

print.sym:          print.map
        MAPSYM —l print.map
```

この例では，５つの目的ファイル作成のための処理が定義されています．各ファイルは，最低１つの関連ファイルと１つのコマンドを持ちます．作業記述は，目的ファイルの作成順序で入力します．例のように，必要な場合は，〝print.exe〟の前に〝startup.obj〟と〝print.obj〟が調べられます．

5.1.3 MAKE の起動

プログラムメインテナ：MAKE を起動するコマンドラインの形式はつぎのとおりです．

```
MAKE 〈メイクファイル名〉
```

〈メイクファイル名〉は，あらかじめ作成しておいたメイクファイルの名前です．メイクファイルの名前は，慣例として，それで開発するプログラムと同じ名前をつけます（拡張子はありません）．メイクファイル名は自由につけることができますが，混乱を防ぐために，この慣例に従うのが望ましいでしょう．

MAKE は起動されると，作業記述を順に調べます．指定された目的ファイルの日付がその関連ファイルの日付より遅れていたり，該当する目的ファイルがなかった場合，MAKE は指定されたコマンドを表示して実行します．そうでない場合は，次の作業記述に飛びます．

指定したメイクファイル，または作業記述中の関連ファイルが見つからなかったときは，MAKE はつぎのようなメッセージを表示します．

```
make:〈メイクファイル名〉—file not found
```

MAKEがコマンドを実行するとき，MAKEの起動時と同じ環境変数を使用します．つまり，PATHコマンドなどによって設定された環境変数が，これらのコマンドの場合も使用できます．

例：

```
make   test
```

この例では，MAKEが〝test〞という名前で作成されたメイクファイルからその命令を取り出します．

5.2 プログラム保守の例

MAKEはプログラムの開発に，特に便利です．MAKEは，ソースプログラムの修正に伴う一連の処理を自動的に行い，修正された実行可能プログラムを素早く作成します．

つぎに示すメイクファイルは，実行可能プログラム〝test.exe〞の開発に使用するものです．ソースファイル名は〝test.asm〞です．この〝test.asm〞は，〝math.lib〞というライブラリファイル内のルーチンを使用します．〝test.asm〞を修正するたびに，そのアセンブルリスト，クロスリファレンスリストの作成，アセンブルされたファイルのライブラリとのリンクが必要となり，さらにSYMDEBで用いるためのシンボルファイルを作成しなければなりません．メイクファイル〝test〞に書かれた作業記述と，それを実行する処理過程をつぎに示します．

```
test.obj:         test.asm
          MASM  test, test, test, test

test.ref:         test.crf
          CREF  test, test

test.exe:         test.obj  ¥lib¥math.lib
          LINK  test, test, test/map,  ¥lib¥math

test.sym:         test.map
          MAPSYM  -l test.map
```

各行は，〝test.obj〞，〝test.ref〞，〝test.exe〞，〝test.sym〞の4つの目的ファイルを作成するために実行する処理を定義しています．

各々のファイルは，少なくとも1つの関連ファイルと，1つのコマンドを持っています．作業記述は，目的ファイル作成の順序で入力されます．例に示したように，〝test.sym〞はLINKの作成する〝test.map〞に，〝test.exe〞はMASMの作成する〝test.crf〞に各々依存してい

ます．

ユーザーは，あらかじめエディタによって，メイクファイル〝test″とソースファイル〝test．asm″を作成します．次に，このメイクファイル名を指定してMAKEを起動します．このときのコマンドラインは，つぎのようになります．

```
MAKE  test
```

MAKEが行う処理の，各ステップをつぎに示します．

1. 〝test．asm″の修正日付と〝test．exe″の日付を比較します．〝test．exe″の日付が遅れている（または存在しない）場合，MAKEは，つぎのコマンドを実行します．

   ```
   MASM test, test, test, test
   ```

   それ以外の場合は，MAKEは，次の作業記述へ進みます．

2. 〝test．ref″と〝test．crf″の日付の比較．〝test．ref″の日付が遅れている場合，MAKEはつぎのコマンドを実行します．

   ```
   CREF test, test
   ```

3. 〝test．exe″と〝test，obj″の日付，およびライブラリファイル〝math．lib″との比較．〝test．exe″の日付がどちらかのファイルと比較して遅れている場合，MAKEはつぎのコマンドを実行します．

   ```
   LINK test, test, test／map,  ¥lib¥math. lib
   ```

4. 〝test．sym″と〝test．map″の日付を比較します．日付が遅れている場合は，つぎに示すコマンドを実行します．

   ```
   MAPSYM －l test. map
   ```

〝test．asm″が最初に作成されたときは，MAKEはすべてのコマンドを実行します．これは，メイクファイル中のすべての目的ファイルがないためです．どの関連ファイルも変更しないでMAKEを再度起動させると，MAKEはすべてのコマンドを飛び越します．ライブラリファイル〝math．lib″を変更してその他の変更を行わないと，MAKEはLINKコマンドを実行します．これは，〝test．exe″の日付が〝math．lib″と比較して遅れているためです．MAKEは，同様にシンボルマップファイルも変更しますが，これは〝test．map″がLINKによって作成されるためです．

MAKEはこのように，ファイルの作成日付を判断しながら処理を行います．

索　引

タ

ナ

ハ

NEC

